Express Mail Label
No. EV 331533572US

Attorney Docket No.
14406US02

APPENDIX A

Title : Radio Frequency Local Area Network

Inventors: Meier, et al.

Attorney Docket No. 14406US02

```
                                                    b
                      i                             b
                                                    b
m m mm    eeee       ii      eeee    r rrr         b bbb
mm m  m  e    e       i     e    e   rr   r       bb   b
m  m  m  eeeeee       i     eeeeee  r             b     b
m  m  m  e            i     e       r             b     b
m  m  m  e    e       i     e    e  r             bb   b
m  m  m   eeee       iii     eeee   r             b bbb
```

```
                        d              ll                                  d
                        d               l                                  d
                        d               l                                  d
m m mm    oooo      ddd d   u    u      l     eeee     ssss             ddd d    oooo     cccc
mm m  m  o    o   d   dd   u    u      l     e    e   s    s           d   dd   o    o   c
c
m  m  m  o    o   d    d   u  - u      l     eeeeee    ss              d    d   o    o   c
m  m  m  o    o   d    d   u    u      l     e           ss            d    d   o    o   c
m  m  m  o    o   d   dd   u   uu      l     e    e   s    s     ..    d   dd   o    o   c
c
m  m  m   oooo     ddd d    uuu u      lll     eeee     ssss      ..      ddd d    oooo     cccc
```

```
                                                    b
                      i                             b
                                                    b
m m mm    eeee       ii      eeee    r rrr         b bbb
mm m  m  e    e       i     e    e   rr   r       bb   b
m  m  m  eeeeee       i     eeeeee  r             b     b
m  m  m  e            i     e       r             b     b
m  m  m  e    e       i     e    e  r             bb   b
m  m  m   eeee       iii     eeee   r             b bbb
```

I. MAC layer files.

- listen-before-talk algorithm:

  lbt.h
  lbt.c

- common routines:

  mac.h
  mac.c

- receive buffers:

  macrxbuf.h
  macrxbuf.c

- unicast receive routines:

  macrxsm.c

- tranmit buffers:

  mactxptr.h
  mactxptr.c

- unicast transmit state machine:

  mactxsm.c

- unicast transmit and receive sequence control cache

  mcbtbl.h
  mcbtbl.c

- low-level transmit/recieve files:

  common.h
  rc3250.h
  scc302.h
  system.h
  vectors.h
  hdlcdriv.h
  internal.h
  io.h
  sstcode.c
  sstphy.c

II. Bridge layer (spanning tree) files.

- address server routines:

  addrsrv.c
  addrsrv.h

- address server task:

  asrvtask.c

- address server task interface:

  asrvtifc.c
  asrvtifc.h

- attached bridge with a zero root priority:

  bb_a.c

- unattached bridge with a zero root priority:

  bb_u.c

- common routines:

  bridge.c

- detached node list:

  detlist.c
  detlist.h

- pending message container files:

  pendmsg.c
  pendmsg.h
  pmq.c
  pmq.h

- attached bridge with a non-zero root priority:

  r_a.c

- unattached bridge with a non-zero root priority:

  r_u.c

- RARP (reverse address resolution packet) routing table:

  rarptbl.c
  rarptbl.h

- packet routing table:

  routetbl.c
  routetbl.h

- bridge function prototypes:

  bridge.pro

III. SST system support routines and header files.

- error codes:

  errcodes.h
  arperr.h
  brgerr.h
  macerr.h
  llcerr.h
  nnmperr.h

- bridge layer system parameters and structures:

  bridge.h

- SST system parameters:

  sstsys.h

- system configuration records (used by the SST task manager):

  sysctl.h

- generic bubble sort routine:

  bublsort.c
  bublsort.h

- bridge layer transmission queue:

  brgtxq.h
  brgtxq.c

- SST system event and data buffer management functions:

  bufpool.h - header file for all buffer functions.
  bufbrg.c - bridge layer specific buffer functions.
  bufcoals.c - buffer coalesce functions.
  bufcopy.c - buffer copy functions.
  bufcre.c - buffer pool create functions.
  bufdisp.c - buffer display function.
  bufenter.c - data entry iterator functions.
  bufevbuf.c - event buffer pointer functions.
  bufhold.c - buffer hold functions.
  bufiter.c - data iterator functions.
  bufllc.c - llc layer specific functions.
  bufmac.c - mac layer specific functions.
  bufpool.c - buffer pool initialization, post, get and release functions.
  bufpostw.c - buffer post and wait function.
  bufptr.c - event union pointer functions.
  bufrepl.c - buffer replicate functions.

- SST device identifier functions:

  deviceid.h
  deviceid.c

- generic integer set functions:

  intset.h
  intset.c

- millisecond timer functions:

  mstimer.h
  mstimer.c

- system millisecond timer task

  mtmrtask.c

- network format to integer conversion functions:

  ntoi.h
  ntoi.c

- generic queue/stack/list implementaion:

  qsl.h
  qsl.c
  qsliter.c - queue/list iterator functions.
  qslpop.c   - stack "pop" function.
  qslret.c   - queue return (i.e. to the head of the queue) function.

- SST bridge layer in range node list:

  rangelst.h
  rangelst.c

- root sequence function:

  rootseq.h
  rootseq.c

- AMX task create functions:

  runproc.h
  runp.c
  runproc.c

- RB4030/RB4020 switch functions:

  sw4030.c

- AMX system timer interface functions:

  timer.h
  timer.c

- timer list functions:

  tmrlist.h
  tmrlist.c

IV. SST protocol stack for a non-Norand device (ACE):

- initialization and event handler:

  ace.h
  ace.c

- ACE terminal MAC layer functions:

  acemac.c

- terminal bridge layer address cache:

  adrcache.c
  adrcache.h

- SST MAC for bus communications to an ACE entity:

  busmac.c
  busmac.h

- ACE higher layer interface functions:

  llcifc.c

- example LLC applications:

  listen.c
  send.c

- dummy LLC layer:

  llc.c
  llc.h

- LLC layer interface definition:

  llcifc.h

- RC3250/RB4020/RB4030 task manager:

  sstmgr.c

- example EEPROM system configuration:

  stbase.c
  st3lisw.c

- terminal bridge layer:

  t_a.c - attached terminal node.
  t_bridge.c
  t_u.c - unattached terminal node.

V. Network management protocol modules.

- nnmp client protocol.

  nnmp.h
  nnmp.c

- client application interface.

  t_nnmp.h
  nnmpifc.h
  nnmpifc.c

- example client application.

  nnmpapp.c

Microfiche Appendix B
Appendix B

Title : Radio Frequency Local Area Network

Inventors: Ronald L. Mahany et al.

Attorney Docket No. 14406US02

Express Mail Label
No. EV 331533572US

Attorney Docket No.
14406US02

APPENDIX C

Title : Radio Frequency Local Area Network

Inventors: Meier, et al.

Attorney Docket No. 14406US02

SST NETWORK ARCHITECTURE

Network Overview.

The SST system implements a hierarchical radio frequency network of, possibly mobile, terminals used primarily for online data entry and occasionally for batch file transfers. The network is characterized by sporadic data traffic over multiple-hop data paths consisting of RS485 or ethernet wired links and single-channel direct sequenced spread spectrum radio links. The network architecture is complicated by moving nodes, hidden nodes, sleeping nodes, transient radio links, and unidirectional radio links.

The SST network consists of the following types of devices:

Host - A host computer which communicates with terminals in the SST network. A host can be viewed as the network addressable software entity which interfaces the SST network to the physical host computer.

Controller - A gateway which passes messages from a host port to the SST network; and which passes messages from the network to a host port. The host port (directly or indirectly) provides a link between the controller and a host computer to which terminals are logically attached. Each autonomous network must have at least one controller device.

Base station - An interior node which is used to extend the range of a controller node. Base-station-to-controller or base-station- to-base-station links can be wired or radio.

Terminal - Norand hand-held computer, printer, etc. A terminal can be viewed as the network addressable software entity which interfaces the SST network to the physical device.

The devices are, logically, organized as **nodes** in an (optimal) **spanning tree**, with a root node in a controller device, internal nodes in base stations or controllers on branches of the tree, and terminal nodes as (possibly mobile) leaves on the tree. With the exception of the root node, each (child) node is connected by a single logical link to a parent node. Like a sink tree, nodes closer to the root of the spanning tree are said to be "**downstream**" from nodes which are further away. Conversely, all nodes are "**upstream**" from the root. Packets are only sent along branches (i.e. logical links) of the spanning tree. Nodes in the network use a "**backward learning**" technique to route packets along branches of the spanning tree.

Devices in the spanning tree are logically categorized as one of the following three node types:

Root (or root bridge) - A controller device which functions as the **root bridge** of the network spanning tree. The spanning tree has a single root node. Initially, all controllers are **root candidates**. One, and only one, root node is determined for each autonomous network by using a priority-based root selection algorithm.

Bridge - An internal node in the spanning tree which is used to "bridge" terminal nodes together into an interconnected network. Note that the root node is a bridge and the term "bridge" may be used to refer to all non-terminal nodes or all non-terminal nodes except the root, depending on the context. A bridge node consists of a **network interface point** and a **routing function.**

Terminal - a leaf node in the spanning tree. A terminal node can be viewed as the software entity that terminates a branch in the spanning tree. A terminal node consists of a **netw rk interface point** and one or more **terminal access points.**

A **controller device** contains a terminal node(s) and a bridge node. The bridge node is the root node if the controller is functioning as the root bridge.

A **base station** contains a bridge node. A base station does not contain a terminal node.

A **terminal device** contains a terminal node.

Bridging entity refers to a bridge node or to the **network interface point** in a terminal device.

Network interface point refers to the single network addressable entity which must exist in all nodes. A network interface point is equivalent to the software entity which is used to interface the SST network to a device or bridging node. Note that a controller device connected to a host computer will have a network interface point which references the host computer and a second discrete network interface point which references the bridging node in the controller. Each network interface point is identified by a unique network address. Unless otherwise specified, this document uses "network address" or simply "address" to refer to the identifier of a network interface point. Note that multiple network interface points may be referenced with multicast and broadcast addresses.

Terminal access point refers to a higher layer access point into the network. A terminal access point is defined by the concatenation of the network interface point address and the terminal access point identifier. A terminal device or controller device can have multiple terminal access points.

A **logical port** is defined by a physical port and a network interface point. This implies that a single device may have more than one physical port with the same network address. In this document "port" refers to a logical port.

Figure 1.1 illustrates the relationship between devices, nodes, terminal access points (TAP), network interface points (NIP) and network routing functions (NRF). The controller device has two network interface points. As an example, the NIP in the controller's terminal node is equivalent to the software entity which interfaces to a host computer. The two terminal access points attached to that NIP identify discrete applications (i.e. terminal emulation and file transfer applications directed to the host computer).

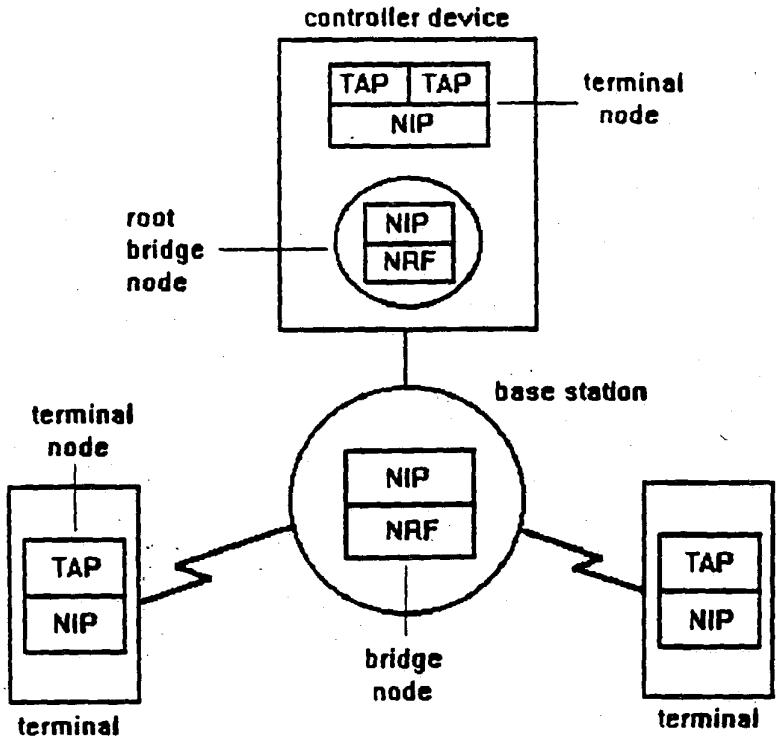

figure 1.1

Basic network requirements:

- Wired or wireless node connections.
- Network layer transparency.
- Dynamic/automatic network routing configuration.
- Terminal mobility. Terminals should be able to move about the radio network without losing a data-link connection.
- Ability to accommodate sleeping terminals.
- Ability to locate terminals quickly.
- Built-in redundancy. Lost nodes should have minimal impact on the network.
- Physical link independence. Higher layer protocols must be consistent across heterogeneous physical links.

Network Layers - Overview.

The SST network software is functionally layered as follows:

Medium Access Control (MAC).

The MAC layer is responsible for providing reliable transmission between ports on any two devices in the network (i.e. terminal-to-base-station). The MAC has a **channel access control** component and a **link control** component. The two components are equivalent to the ISO media access control and data link control sublayers, respectively.

The link control component facilitates reliable point-to-point frame transfers in the absence of collision detection and in the presence of errors.

Several channel access control algorithms are used to regulate access to the communications channel. The algorithms are link-type dependent.

A **p-persistent CSMA/CA** (carrier sense multiple access with collision avoidance) protocol is used to gain access to an **RS485 LAN**. The collision avoidance scheme gives channel access priority to the recipient of a unicast frame.

On lightly loaded spread spectrum radio links, a **non-persistent CSMA** algorithm is used to gain access to the communications channel. Under moderate to heavy channel utilization, an **LBT/BP** (listen-before-talk with busy pulse) algorithm is used to gain access to the channel and minimize the effect of hidden nodes.

A **polling protocol** is used to restrict contention to **request-for-poll (RFP)** frames, thus minimizing contention for data frames.

The channel access control algorithms incorporate a random backoff algorithm to prevent deadlock and instability in contention situations.

Bridging Layer.

The bridging layer has several functions:

1) The bridging layer uses a "HELLO protocol" to organize nodes in the network into an optimal spanning tree rooted at the root bridge. The spanning tree is used to prevent loops in the topology. Interior branches of the spanning tree are relatively stable (i.e. controllers and relay stations do not move often). Terminals, which are leaves on the spanning tree, may become unattached, and must be reattached, frequently.

2) The bridging layer routes packets from terminals to the host, from the host to terminals, and from terminals to terminals along branches of the spanning tree.

3) The bridging layer provides a service for storing packets for SLEEPING terminals. Packets which cannot be delivered immediately can be saved by the bridging entity in a parent node for one or more HELLO times.

4) The bridging layer propagates lost node information throughout the spanning tree.

5) The bridging layer maintains the spanning tree links.

6) The bridging layer distributes network interface addresses.

7) The bridging layer organizes nodes into logical coverage areas on radio channels.

Data-link (transport) layer.

The data-link layer provides an end-to-end data path between data-link access points in any two nodes in the network. The data-link layer provides a connection-oriented reliable service and a connectionless unreliable service. The reliable service detects and discards duplicate packets and retransmits lost packets. The unreliable service provides a datagram facility for upper layer protocols which provide a reliable end-to-end data path.

The data-link layer provides ISO layer 2 services for terminal-to-host application sessions which run on top of an end-to-end terminal-to-host transport protocol. However, the data-link layer provides transport (ISO layer 4) services for sessions contained within the SST network.

Higher Layers.

For terminal-to-terminal sessions contained within the SST network, the data-link layer provides transport layer services and no additional network or transport layer is required. In this case, the MAC, bridging, and data-link layers discussed above can be viewed as a data-link layer, a network layer, and a transport layer, respectively. For terminal-to-host-application sessions, higher ISO layers exist on top of the SST data-link layer and must be implemented in the terminal and host computer, as required. This document does not define (or restrict) those layers. This document does discuss a fast-connect VMTP-like transport protocol which is used for transient internal terminal-to-terminal sessions.

Network address requirements.

MAC frames contain a hop destination and hop source address in the MAC header.

Bridging packets contain an end-to-end destination and source address in the bridging header.

Data-link headers contain source and destination access point identifiers. A data-link connection is defined by the concatenation of the bridging layer source and destination address pairs and the destination and source data-link access points. One end of a connection is equivalent to a terminal access point and is specified as <access_point>@<network_address>, where aliases can be used for both.

MAC and bridging addresses are consistent and have the same format.

All devices must have either a unique **long identifier** which is programmed into the device at the factory and/or an **alias** which is typically entered by the user or is well-known. The long address/alias binds to a short **network address**, obtained from an address server in the root node. A network address uniquely identifies the **network interface point** in each node.

The **network interface point**, in each node, obtains a **network address** from an address server in the root, which uniquely identifies the node. The network interface point passes the network address to the MAC entity attached to each port on a device. Short addresses are used to minimize packet sizes.

Network addresses consist of 2 parts:

- a **node type**. Node types are well-known. Some node types are used for multicast messages (i.e. all bridges).

- a unique, multicast, or broadcast **node identifier**.

A node-type identifier of all 1's is used to specify all node types.

The node identifier part of a root address is all 0's.

A node identifier of all 1's is used to specify any node of the specified type, and is the default node identifier, used before a unique node identifier is obtained.

In addition, to source and destination addresses, each network packet contains a **spanning tree identifier** in the MAC header.

A default spanning tree identifier is well-known by all nodes.

A non-default spanning tree identifier can be entered into the root node (i.e. by a network administrator) and advertised to all other nodes in HELLO.response packets. The list of non-default spanning trees to which other nodes can attach must be entered into each node.

Network address allocation.

The network node identifier of a root node is always all 0's and is well-known. All other nodes must obtain a unique network node identifier from a Reverse Address Resolution Protocol (RARP) server in the root node. A node identifier of all 1's is used until a unique identifier is obtained. To get a unique identifier, a node must send a RARP.request packet to the RARP server. The packet contains the requesting node's unique **long identifier** and/or an **alias** for the long identifier. A **network address** is returned to the requesting node in a RARP.response packet.

Nodes must obtain a (new) network address whenever a new root node is discovered and whenever an ADDRESS_TIMEOUT inactivity period expires without the node receiving a packet from the bridging entity in the root. (A node can prevent its address from expiring by sending an empty **attach.request** packet to the root.)

The address server in the root associates an age factor with each allocated network address. The age factor is incremented each time an ADDRESS_TIMER expires. The age factor, associated with an address, is reset to 0 whenever the root receives a packet from the address. An address is available for use by a requesting node if it has never been used or if it has been inactive for a MAX_ADDRESS_LIFE time period.

MAX_ADDRESS_LIFE must be much larger than ADDRESS_TIMEOUT to ensure that an address is not in use by any node when it becomes available for another node. If the root receives a RARP.request packet from a source for which an entry exists in the address queue, the root simply resets the age factor to zero and returns the old address.

Bridging layer theory and implementation notes.

The bridging layer organizes nodes into an optimal spanning tree with a single root bridge at the root of the tree. (Note that the spanning tree identifier allows more than one logical tree to exist in the same coverage area.) Spanning tree organization is facilitated with a HELLO protocol which selects a root node and enables nodes to determine the shortest path to the root before attaching to the spanning tree. All messages are routed along branches of the spanning tree. Restricting each node to a single parent guarantees that there will be no loops in the logical topology.

Figures 2.1 and 2.2 illustrate how physical devices are organized into logical nodes in a spanning tree. Figure 2.1 depicts devices and the physical communication links. Figure 2.2 depicts the same devices organized as nodes on branches of a spanning tree. The root node in fig. 2.2 is labeled with an R.

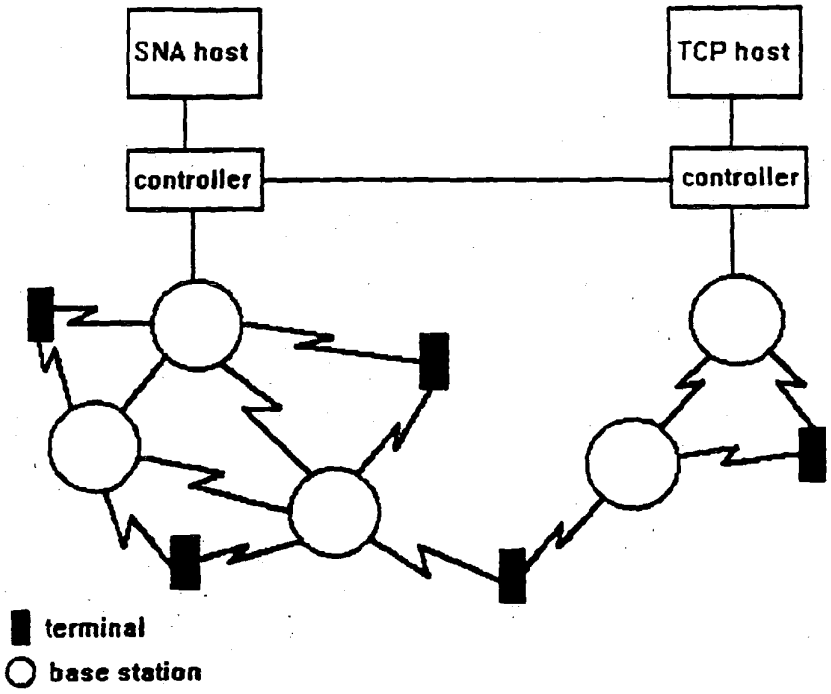

figure 2.1

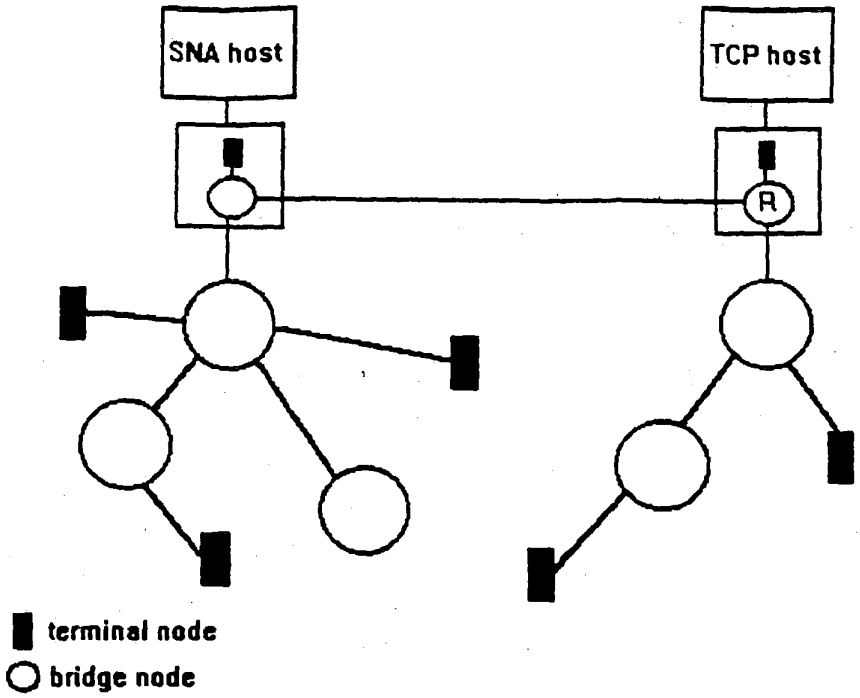

figure 2.2

Spanning tree organization.

Nodes in the network are generally categorized as ATTACHED or UNATTACHED. Initially, only the root is attached. A single controller may be designated as the root, or multiple root candidates (i.e. controllers) may negotiate to determine which node is the root.

Attached bridge nodes and root candidates transmit HELLO.response packets at calculated intervals. The HELLO.response packets include:

- the source address.

- a broadcast destination address.

- the distance (cost) to the root.

The incremental portion of the distance between a node and its parent is primarily a function of the physical link type (i.e. ethernet, RS485, or radio). On radio links, bridging connections are biased toward the link with the best signal strength. Signal strength is not a factor in the cumulative path distance. The distance component is intended to bias path selection toward high-speed (i.e. wired) connections.

- a "seed" value used to calculate the time of the next HELLO.response packet.

- a hello slot displacement. The displacement specifies the displacement of the actual hello slot time from the calculated hello slot time or to indicate that the hello time was not calculated (i.e. was unscheduled).

- a spanning tree identifier (LAN ID).

- the priority of the root node (or root candidate).

- the long, unique device identifier of the root node (or root candidate).

- a root node sequence number, used to distinguish between multiple occurrences of the spanning tree with the same root node.

Attached nodes must forget their network address and return to the UNATTACHED state whenever a HELLO.response packet is received with a new root node identifier or sequence number.

Optional parameters:

- descendent count.

- a pending message list.

Pending message lists consist of 0 or more network addresses for SLEEPING terminals. Pending messages for terminals are stored in the parent node.

- a detached-node list.

Detached-node lists contain the addresses of nodes which have detached from the spanning tree. An internal node learns which entries should be in its list from DETACH packets which are broadcast by internal nodes when a child is lost. Entries are included in HELLO.response packets for DETACH_MSG_LIFE hello times.

Attached nodes broadcast short HELLO.response packets immediately if they receive a HELLO.request packet with a global destination address; otherwise, attached nodes will only broadcast HELLO.response packets at calculated time intervals. Short HELLO.response packets are sent independently of regular HELLO.response packets and do not affect regular hello timing.

Unattached nodes (i.e. without a parent in the spanning tree) are, initially, in an UNATTACHED state. During the unattached state, a node learns which attached bridge is closest to the root node by listening to HELLO.response packets. After the learning period expires an unattached node sends an ATTACH.request packet to the attached bridge node closest to the root. (Nodes without a network address must first send a RARP.request packet to the root to obtain a network address.) The attached node adopts the unattached node as a child by acknowledging the ATTACH.request packet and forwarding it onto the root node. The root node returns the request as an end-to-end ATTACH.response packet. If the newly attached node is a bridge, it calculates its distance to the root, by adding its link distance to the total distance of its new parent, and begins to transmit HELLO.response packets.

The end-to-end ATTACH.request functions as a discovery packet, and enables nodes in the path to the root node to quickly learn the address of the source node.

The UNATTACHED learning state ends after HELLO_RETRY hello time slots if HELLO.response packets have been received from at least one node. If no HELLO.response packets have been received the listening node waits (i.e. sleeps) and retries later.

An attached node may respond to a HELLO.response packet from a node other than its parent (i.e. with an ATTACH.request packet) if the difference in the hop count specified in the HELLO.response packet exceeds a CHANGE_THRESHOLD level.

Unattached nodes may broadcast a **global** HELLO.request with a multicast bridge destination address to solicit short HELLO.response packets from attached bridges. The net effect is that the UNATTACHED state may (optionally) be shortened. (Note that only attached bridges or the root may respond to an ATTACH.request packet.) Normally, this facility is reserved for terminals with transactions in progress.

ATTACH.request packets may contain a descendants list, so that an internal node may attach itself and the subtree under it to a bridge node **closer** to the root.

ATTACH.request packets contain a "delivery service type" field, which indicates that a terminal (i.e. which sent the request) may be SLEEPING, and a "maximum stored message count" field. The bridging entity in the parent of a SLEEPING terminal will temporarily store messages, for later delivery, if that service is requested. The bridging entity in a parent node will store pending messages until 1) the message is delivered, or 2) "maximum stored message count" hello times have expired.

Data-link layer data can be piggybacked on an ATTACH.request packet from a terminal.

Attachment criteria.

On wired links, the weighted distance is the only criteria for choosing a parent.

On radio links, a parent is chosen based on the following criteria:

1) The signal strength must exceed a minimum threshold value.
2) If two potential parent nodes are at a different distance from the root, the one with the least distance is chosen.
3) If two potential parent nodes are at the same distance, the node with the best signal strength is chosen.
4) If two potential parent nodes are at the same distance and have the same signal strength, then the node with the highest priority is chosen.
5) If the distance, signal strength and priority are equal, then the node with the highest long ID or alias is chosen.

The intent of the above criteria is to create stable disjoint **logical coverage areas** in the presence of physically overlapping coverage areas. Ideally, all radio terminals in a coverage area will be attached to a single bridge node.

The concept of logical coverage areas is illustrated in the figure below.

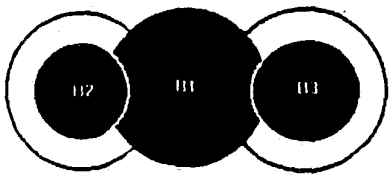

figure 3.1

Three radio bridge nodes, B1, B2, and B3, with network addresses of 1, 2, and 3 are depicted in figure 3.1. All three nodes are assumed to be at the same distance from the root. The circles around each node denote physical coverage areas. The dark shaded areas denote strong signal coverage areas. The logical coverage of B1 is depicted by its dark and light shaded areas. The logical coverage areas of B2 and B3 are depicted by the associated dark shaded and non-shaded areas. Note that the dark shaded area of B1 would logically hide the dark shaded area of B2 or B3 if the areas physically overlapped. Also note that the logical coverage area of a fourth bridge with the same distance and a higher address, say B4, could be completely hidden behind one of the three bridge nodes, if B4 and one of the other bridges were located side-by-side.

The concept of disjoint logical coverage areas is especially important when radio bridge nodes are placed in close proximity to provide redundant coverage (i.e. for protection against a failure). The MAC entity in one of the bridge nodes can efficiently regulate access to the channel (i.e. by queuing terminals for polling) without coordination with other co-located bridge nodes.

Bridging layer routing.

All packets are routed along branches of the spanning tree. Bridges "learn" the address of terminals by monitoring traffic from terminals (i.e. to the root). When a bridge receives a packet directed toward the root (i.e. moving **downstream**), the bridge creates or updates an entry in its routing table for the terminal. The entry includes the terminal address, and the address of the bridge which sent the packet (i.e. the hop source address). When a bridge receives an **upstream** packet (i.e. moving from the root, destined for a terminal) the packet is forwarded to the upstream node which is specified in the routing entry for the destination. Upstream packets are discarded whenever a routing entry does not exist. Downstream packets (i.e. from a terminal to the root) are simply forwarded to the next downstream node (i.e. the parent) in the branch of the spanning tree. **No explicit routing is required for downstream traffic** since the route is defined by the structure of the spanning tree. A packet travels downstream until a node is reached which has an entry in its routing table for the destination address. The packet is then explicitly routed upstream until it reaches its destination. Thus, **terminal-to-terminal communications is accomplished by routing all traffic through the nearest common ancestor** of both terminals. In the worst case, the root is the nearest common ancestor. An address resolution server in the root node facilitates terminal-to-terminal communications (see below).

For example, in figure 3.1, if terminal E sends a packet to terminal B, the packet will follow downstream hops from E to 3, 3 to 2, 2 to 1, and 1 to R, where R is the root node. Routing tables are not required for the downstream hops. The routing function at R has an entry for B in its routing table which specifies B as the first upstream hop to B. Therefore the packet is explicitly routed upstream from R to B.

As a second example, if terminal D sends a packet to terminal E, the packet will follow one downstream hop from D to 2. The routing function at 2 has an entry for E in its routing table which specifies 3 as the first upstream hop to E. The packet is routed upstream to 3. An entry in the routing table at 3 specifies E as the first upstream hop to E, and the packet is routed from 3 to E.

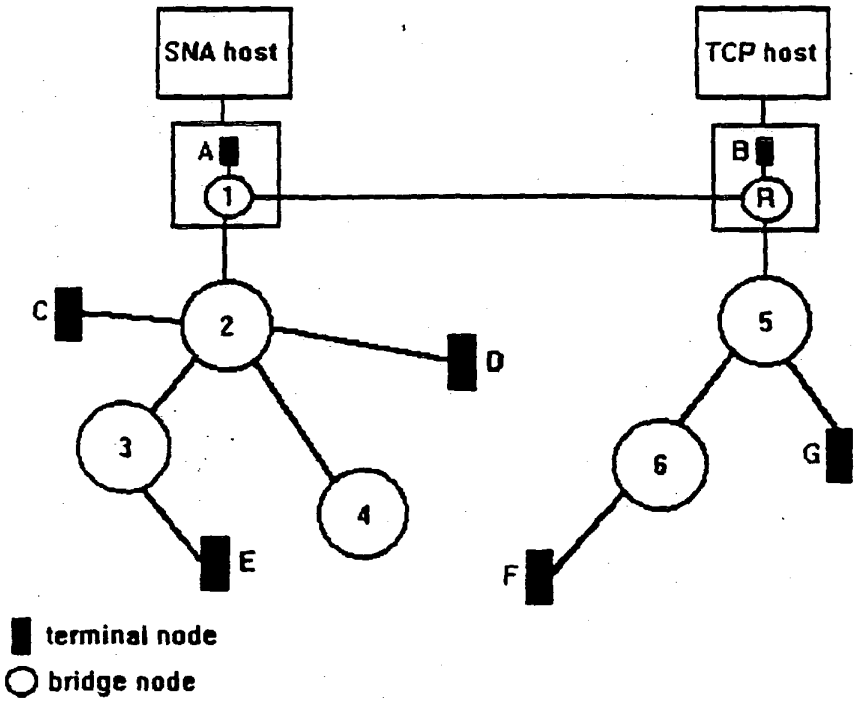

figure 3.4

As an extension to the routing algorithm described above, terminals may optionally cache the addresses of neighbors in a separate direct route table. If a terminal has a message for a destination listed in its direct route table, it may transmit it directly to the source node. Note that the packet may not follow a branch of the spanning tree. Direct route table entries must be aged relatively quickly. If a direct transmission fails, the entry in the direct route table is discarded and the packet is simply forwarded downstream to the root. The header format field in the bridge header must be set to point-to-point for directly transmitted packets. Direct routing has obvious advantages; however it forces terminals to maintain additional MAC layer state information.

As an example of direct routing, in figure 3.4, terminal E can route packets directly to F, if E has an entry for F in its direct routing table.

Note that the direct routing table in a node is built by listening to traffic directed to other nodes. If the MAC layer screens such traffic from the bridging layer, the direct routing table must be built by the MAC layer.

Dynamic Changes in the Spanning Tree.

Paths in the spanning tree can change for a number of reasons:

1) Any node may select a new path to the root whenever a better path is found. For example, a better path might be one where the distance of a node's parent from the root is CHANGE_THRESHOLD greater than the distance in a HELLO.response packet from another node. Note that CHANGE_THRESHOLD can be as small as 1. All else being equal, a node on a radio channel should always choose, as its parent, the node with the best signal strength. A node can move its entire subtree by including a decedents list in the ATTACH.request packet sent to the new parent. Rapidly moving terminals can cache a short list of alternate parents. Periodically, SLEEPING terminals, must stay awake for 1+ HELLO times to discover changes (i.e. shorter paths) in the network topology.

2) A parent node detaches the subtree rooted at a child node, whenever a message cannot be delivered to the child. This occurs when the MAC layer in a parent node fails to deliver a unicast bridging layer packet to a child node. In addition, the bridging entity in a parent node can retain messages for a child terminal node. Terminals request the saved messages by sending a request packet to the parent with an "**awake time**" parameter. If the message is not requested and delivered after "**store message count**" hello times, then the terminal is detached. If the detached node is a bridge node, the parent node will send a DETACH.request packet, to the root node, which contains a decedents list that defines the lost subtree. If the detached child is a terminal, the parent will flood a DETACH.request throughout all branches of the spanning tree using a **reliable flooding mechanism.**

The detached node information which is broadcast in flooded DETACH.request packets is added to the "detached node set" maintained in each bridge node. The detached node set is copied into the detached node list in the bridge's HELLO.response packets. The entries in the detached node list are discarded after MAX_HELLO_LOST+1 HELLO times.

3) A child node goes into the UNATTACHED state whenever its MAC layer fails to deliver a message to its parent. If the child node is a bridge, it must continue to broadcast scheduled HELLO.response packets with an **infinite distance** for MAX_HELLO_LOST+1 times. If the child node is a terminal, it may solicit short HELLO.response packets to shorten the UNATTACHED state. After the UNATTACHED learning state has expired the node reattaches by transmitting an ATTACH.request to the bridge node closest to the root.

4) If a node in an ATTACHED state receives a DETACH packet or a HELLO.response packet with its network address in the detached list, it must enter the UNATTACHED state and reattach to the spanning tree. Note that it may not actually be unattached. The node can shorten the UNATTACHED state by soliciting short HELLO.response packets. After reattaching, the node must remain in a HOLD_DOWN state for MAX_HELLO_LOST+1 hello times. During the HOLD_DOWN state, the node ignores its address in DETACH packet and HELLO.response packet detached lists. After the HOLD_DOWN period expires the node must send a second ATTACH.request to the root, to insure that it is still attached.

5) Entries in routing tables are aged. When routing table space for a new entry is required, either an unused entry or the oldest (i.e. least recently used) entry is selected. If a used entry is selected, then the old information is simply discarded. The aging factor associated with each table entry is reset to 0 each time a new packet from the associated node arrives. A node must periodically send an ATTACH.request packet to the root node to maintain its path in the spanning tree. The ATTACH.request can be piggybacked on a higher-layer data packet if the node is busy. If the root node is on the path to the destination of the data packet then the overhead required to keep the node's address alive is minimal.

6) A SLEEPING node must wake up and enter an ATTACHED listen state whenever a threshold number (i.e. 1 or 2) of HELLO.response packets, from its parent, are missed. The state ends when the node receives a data or HELLO.response packet from its parent. The node enters the UNATTACHED state when a) its address appears in the detached list of a DETACH or HELLO.response packet, or b) a total of MAX_HELLO_LOST consecutive HELLO.response packets are missed.

The time that a node spends in the ATTACHED LISTEN state must be less than the lifetime of detached node information in the network. This insures that a detached node will always enter the UNATTACHED state (i.e. either the node will find its address in a detached node list or the node will miss MAX_HELLO_LOST HELLO.response packets and go into the UNATTACHED state before it sees a "good" HELLO.response packet from its (former) parent.

7) Any node which receives a HELLO.response packet from its parent with an infinite distance immediately enters the UNATTACHED state. If the node is a bridge, it must continue to broadcast HELLO.response packets with an infinite distance for MAX_HELLO_LOST+1 times.

Note that old invalid paths may exist in the spanning tree for a period of time. For example, if a terminal detaches and reattaches to a different branch in the spanning tree, all downstream nodes in the new branch (quickly) "learn" the new path to the terminal. Nodes which were also in the old path change their routing tables and no longer forward packets along the old path. At least one node, the root, must be in both the old and new path. A new path is established as soon as an end-to-end attach request packet from the terminal reaches a node which was also in the old path. Any remaining old path fragment will be disjoint from the new path.

Detach Packet Logic.

A parent node generates a DETACH.request packet whenever it is unable to deliver a message to a child node. The two possible cases are 1) the child is a bridge, and 2) the child is a terminal.

Case 1 - The lost node is a bridge.

When a parent node is unable to deliver a message to a child bridge node, it must send a DETACH.request packet, to the root node, which contains a detached node list that describes the lost subtree. The list will contain all nodes in the routing table of the parent for which the lost bridge was the first upstream hop. All downstream nodes in the path of the DETACH packet must adjust their routing tables by deleting entries which match those in the detached list.

Case 2 - The lost node is a terminal.

Since terminals can be mobile they can be lost often and must be notified quickly. When a parent node is unable to deliver a message to a terminal, it must generate a DETACH.request packet, with the terminal specified in the associated detached node list, and flood the packet throughout all branches of the spanning tree. The packet is forwarded using a reliable broadcast mechanism. The DETACH packet contains a **forward list** which is used to specify which nodes should forward and acknowledge the DETACH.request. Initially, the forward list consists of all bridges which are either children or the parent of the node which generated the packet. Nodes in the forward list acknowledge the DETACH.request, with a DETACH.response, and forward the DETACH.request along all branches of the spanning tree except the branch it was received on, with one exception. A bridge node in the forward list does not forward an entry in the detached list of a DETACH.request if 1) the DETACH.request came from an upstream node, and 2) the upstream node is not the first hop in the routing table entry associated with the entry in the detached list. Upstream bridges, which do not have bridge nodes as children, broadcast the DETACH.request one time (without a forward list).

Note that the destination address used to forward a flooded DETACH.request is global. Therefore, the detached terminal may receive a DETACH.request and quickly learn that it has been detached. All bridge nodes, which receive the DETACH.request, add the detached terminal to their detached node list which is broadcast in HELLO.response packets for MAX_HELLO_LOST+1 times or until the bridge determines the terminal has reattached.

Hello synchronization.

All attached non-terminal nodes broadcast periodic HELLO.response packets at calculated intervals. On the average, the intervals are separated by HELLO_PERIOD seconds. The HELLO.response packet contains a "seed" field used in a well-known randomization algorithm to determine the next hello time for the transmitting node and the next seed. The address of the transmitting node is used as a factor in the algorithm to guarantee randomization. Nodes can execute the algorithm i times to determine the time (and seed) of the i-th hello packet from the transmitter.

After attaching, a bridge chooses a random initial seed and a "hello slot" and broadcasts a hello packet in that slot. The bridge chooses succeeding hello slots by executing the randomization algorithm. If the transmission of a HELLO.response packet is delayed, then the delay is entered into a hello "displacement" field in the packet, so that the calculated time can be accurately derived by a receiver. Cumulative delays are not allowed (i.e. contention delays during the i-th hello transmission do not effect the time of the i+1 hello transmission).

A node initially synchronizes on a HELLO.response packet from its parent. A SLEEPING node can calculate the time of the next expected HELLO.response packet from its parent and can power-down with an active timer interrupt set to wake it just before the HELLO.response packet is transmitted. The bridging entity in a parent node can store a message for a SLEEPING node until the SLEEPING node "requests" the message by notifying its parent that it is awake. A terminal learns that it must request unsolicited saved message by examining the pending message list in the HELLO.response packet. This implementation enables SLEEPING terminals to receive unsolicited messages and relaxes the timing constraints for transaction oriented messages. Retries for pending messages are transmitted in a round-robin order when messages are pending for more than one destination.

Note that a child node, which misses i HELLO.response packets, is able to calculate the time of the i+1 HELLO.response packet.

Data-link/Transport layer theory and implementation notes.

The data-link layer is implemented as an extension of Class 2 Logical Link Control (LLC) as defined in ISO Standard 8802-2.2. The extensions to LLC are 1) an additional unnumbered command frame - SABMX, and 2) 15-bit send and receive sequence numbers. In addition, the implementation must include an adaptive time-out algorithm for retransmissions. Unreliable (type 1) and reliable (type 2) connection-oriented services are provided. The unreliable service is provided for terminals which support a reliable end-to-end transport protocol with a host computer. LLC type 2 provides a reliable end-to-end transport service for long-lived terminal-to-terminal connections within the spanning tree network.

In addition, a fast-connect VMTP-like transport protocol, is used for transient terminal-to-terminal connections. The VMTP-like service is primarily provided for remote procedure calls (RPC), client/server transactions, and short mail messages.

Note that the bridging layer does not provide a reliable end-to-end service and that lost and duplicate packets must be handled by a higher layer. The bridging layer does not fragment packets and packets are normally delivered in sequence.

The interfaces to the next upper (i.e. application) layer include:

handle=CONNECT(destination, . . .)

handle=LISTEN(SSAP, . . .)

SEND(handle, buffer, length, [destination])

DATAGRAM(handle, buffer, length, [destination]);

TRANSACTION(handle, tx_buf, tx_len, rx_buf, max_rx_len,
[IDEMPOTENT, destination)

RECEIVE(handle, buffer, max_length, [destination])

PENDING_MESSAGE(handle, [destination])

DISCONNECT(handle)

Destination fields are formed by concatenating the destination service access point (DSAP) with the destination network address, where aliases are used for both . For example, 3270@HOST1 might designate a 3270 terminal controller application in a controller node. The DSAPs for common applications are well-known. Note that the DSAP can specify a remote terminal application or the access point to a higher layer protocol in a remote node.

The "handle" designates the connection type, and is the connection identifier for LLC connections.

The optional "destination" field in send and receive operations is only used for the VMTP-like interface.

SEND messages require a response.

DATAGRAM messages do not require a response. DATAGRAM is used to send messages to a host which is capable of supporting end-to-end host-to-terminal transport connections.

TRANSACTION is used to send transaction-oriented messages with the VMTP-like facility. An error occurs if a return message is not received in a TRXN_TIME-OUT period. The data-link/transport entity saves response messages and resends the response when a duplicate transaction message is received. In addition, an application can mark a transaction as redoable, by setting the IDEMPOTENT flag ON. In this case, the response message is not saved and the response is regenerated by re-executing the transaction. Note that a response message can be guaranteed in the form of an acknowledgment from a higher layer protocol.

Because the bridging layer provides an unreliable service, the data-link layer is required to detect duplicate packets and retransmit lost packets. Detecting duplicates is facilitated by numbering data-link packets with unambiguous sequence numbers.

Data-link connections.

LLC type 2 connections are established by sending a SABMX control frame to the destination network address. To prevent frames from an old connection from being accepted (i.e. with a sequence number of 0) the node which initiates a connection must insure that at least MAX_PACKET_LIFE time has expired since the last connection before issuing a new CONNECT for the same destination. Because of the required waiting period, Type 2 LLC connections are not ideal for the type of transient "connections" needed to reliably facilitate remote procedure calls, client/server transactions, and sporadic mail messages.

LLC frames are sequenced from 0 to MAX_SEQ. The maximum number of outstanding frames (i.e. transmitted, but not acknowledged) is LLC_WINDOW_SIZE. The default value of LLC_WINDOW_SIZE is relatively small, but the window size may be expanded with an XID frame. Since all frames sent during a connection may not follow the same path, no more than MAX_SEQ frames may be sent in a MAX_PACKET_LIFE time period. A problem can arise when a node successfully transmits a data-link frame to the next downstream hop on a busy path, but loses all acknowledgments. At this point, the node is detached and must quickly reattach to the spanning tree. If the next parent of the node is on a shorter, less busy branch, frames on the new path can easily arrive at the destination while old frames still exist in the old path. MAX_PACKET_LIFE is equal to (MAX_HOPS · XMIT_Q_SIZE · MAX_RETRY_TIME), where MAX_HOPS is the maximum length of a branch of the spanning tree, in hops, XMIT_Q_SIZE is the number of packets which can be queued in each node, and MAX_RETRY_TIME is the maximum time the MAC layer can spend retrying a frame before it is successfully sent. This problem was addressed by increasing the size of the send and receive sequence number fields (i.e. from 7 bits to 15 bits) so that the N(S) and N(R) fields in an information frame can never roll over faster than MAX_PACKET_LIFE time. Note that the spanning tree topology insures that packets will not loop.

VMTP-like connection records are built automatically. A VMTP-like connection record is built (or updated) whenever a VMTP-like transport message is received. The advantage is that an explicit connection request is not required. A VMTP-like connection is half-duplex. (A full-duplex connection at a higher layer can be built with two independent half-duplex VMTP-like connections.) Acknowledgments must be handled by higher layers.

Connections are defined by the concatenated network end-to-end destination and source addresses, and destination and source service access points.

The LLC type 2 data-link entity in a node stores messages for possible retransmission. Note that retransmissions may not always follow the same path (primarily) due to moving terminals and the resulting changes in the spanning tree. For example, the bridging entity in a parent node may disconnect a child after the MAC entity reports a message delivery failure. The child will soon discover that it is detached and will reattach to the spanning tree. Now when the data-link entity (i.e. in the root) resends the message, it will follow the new path.

Data-link message timing and sleeping terminals.

The data-link entity in a terminal calculates a separate time-out for SEND and TRANSACTION operations. Initially, both time-outs are a function of the distance of the terminal from the root node.

A TCP-like algorithm is used to adjust the expected propagation delay for each message type to the end-to-end distance and load, without causing sporadic changes or dramatic swings in time-out values. Messages, which require a response, are retransmitted if twice the expected propagation time expires

before a response is received. SLEEPING terminals can power down for a large percentage of the expected propagation delay before waking up to receive the response message. Note that missed messages may be stored by the bridging entity in a parent node for "store message count" hello times.

The LLC entity at each end of a connection should be notified when a bridging layer path change has occurred. The notification will trigger an RR or data LLC message transmission if unacknowledged messages exist on the connection, or an expected reply has not been received.

Medium Access Control (MAC) theory and implementation notes.

The MAC layer is responsible for providing reliable transmission between any two nodes in the network (i.e. terminal-to-bridge).

Access to the network communications channel is regulated in several ways:

- Nodes are grouped into logical coverage areas associated with a single bridge node.
- CSMA and LBT algorithms are used to gain access to the channel.
- A polling protocol reduces contention for data frames.

IEEE 802.3 media access control is used for ethernet links.

A p-persistent CSMA/CA with ARQ (automatic retry request) protocol is used to gain access to the channel on the RS485 LAN.

A **collision avoidance protocol** is implemented on RS485 LAN links. Bridging layer packets are typically sent in a single MAC layer data frame on both ethernet and RS485 LAN links. Short blocks can be transmitted as soon as an idle channel is detected. Before a long data frame can be transmitted on a wired link a potential transmitter must sense an idle channel, transmit an RFP frame and receive a POLL frame from the receiver. After a data frame is transmitted, the receiver notifies all listening nodes that the channel is free by sending a CLEAR frame.

A simple **return priority mechanism** is implemented by requiring a potential transmitter to sense an idle line for an IDLE_TIME period which exceeds the maximum transmitter/receiver turnaround time. The recipient of a unicast frame "owns" the channel for the turnaround time and can respond without executing the CSMA algorithm. This approach makes response times more deterministic and allows the sender to set response time-outs tightly. Short time-outs allow transmitting nodes to quickly retry out and discover disconnected links.

A CSMA **random backoff algorithm** specifies backoff delays as a function of the CSMA slot time. A CSMA slot is calculated as a function of the worst-case carrier sense ambiguous period. If, for example, in the worst case, it takes a character-time to determine that a frame is in progress then the CSMA slot time is defined to be slightly longer than one character time. The algorithm divides the sense time into p contiguous slots and chooses a number, i, between 1 and p. If the first i slots are idle then the algorithm allows transmission in the i+1 slot. If one of the first i slots is busy, then the device executing the algorithm listens until the channel is idle and re-executes the algorithm.

A **polling protocol**, which is consistent with the collision avoidance protocol used on wired links, is used to gain access to the channel on spread spectrum radio links. The polling protocol reduces contention, in an environment with **hidden terminals**, in several ways.

On radio links, a MAC transmitter fragments a bridging layer packet into short fixed length frames before the packet is sent. The fragments are reassembled by the receiver and are posted to the receiver's

bridging layer if, and only if, all frames in the packet are received. A group of frames which is associated with a single bridging layer packet is called a **bracket**. Fragmentation at the MAC layer allows the MAC entity to used a (shorter) frame size which is suitable for the link error rate without impacting packet sizes at the bridge layer.

On radio links, the polling protocol generally limits contention to RFP frames. Before a bracket of frames can be transmitted on a radio link, a potential transmitter must sense an idle channel, transmit an RFP frame and receive a POLL frame from the receiver. If the receiver is busy it will respond with a wait-for-poll (WFP) frame. The WFP frame positively acknowledges the RFP frame and causes the transmitter to wait for a POLL frame. Nodes are queued for polling in the order in which RFP frames arrive. After the last frame in a bracket is transmitted and successfully received, the receiver sends a CLEAR frame to notify all listening nodes. Since coverage areas tend to be logically disjoint a single bridge can generally determine access to the channel, within its coverage area, for the transmission of all frames except for RFP (and ENQ) frames directed to the bridge.

The MAC entity attached to a radio port must monitor traffic volume on the port. Under lightly loaded conditions a non-persistent **CSMA/CA with ARQ** algorithm is used to gain access to the channel. A node simply listens and transmits an RFP (request-for-poll) frame immediately if the channel is sensed to be idle. If the channel is busy, the node defers until later.

Under moderate to heavy load conditions, the polling protocol incorporates a p-persistent **LBT/BP with ARQ** algorithm to regulate access to the channel. As Kleinrock and Tobagi noted in [1], the throughput performance of a packet radio network can be significantly degraded by as few as 5 to 10 per cent hidden nodes. In [3] Tobagi discussed the use of a busy tone on a separate frequency. The disadvantage of the busy tone solution is that it requires two transmitters and receivers per node. The LBT/BP protocol provides a somewhat equivalent solution with the use of a single frequency. After the first RFP frame, all other frames sent in a sequence of frames associated with a single bracket are sent without sensing the channel. Because of this **return priority mechanism** and because the MAC entity fragments packets into frames of a fixed size (or smaller) the time between POLL frames (and DATA frames) is fixed. The POLL (or DATA) frames in a bracket provide a **busy pulse** which notifies other nodes in the coverage area of either the transmitter or receiver that a conversation is in progress. All nodes are required to sense an idle channel for a time which exceeds the interpoll gap time before transmitting an RFP frame. The LBT/BP protocol insures that an active conversation will be detected if either the active transmitter or receiver are in range (but not necessarily both). Figure 4.1 below further illustrates this point.

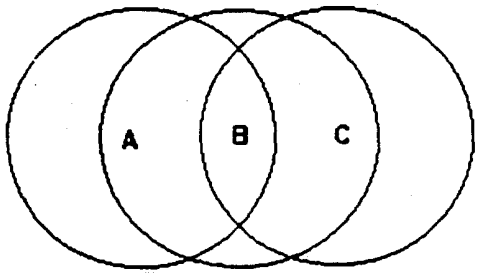

figure 4.1

The example in figure 4.1 depicts a three nodes, A, B, and C, with physical radio coverage areas represented by the circle around each node. Note that B is in the coverage area of both A and C, A is in the coverage area of B but not in the coverage area of C, and C is in the coverage area of B but not in the coverage area of A. Assume that B is polling A to solicit data frames when C first attempts to acquire the channel. Since the LBT algorithm requires C to detect an idle channel for a time which exceeds the

interpoll gap time. C will detect a poll frame from A and will defer from transmitting until the conversation is finished.

An **LBT random backoff algorithm** for radio links specifies backoff delays as a function of the CSMA slot time and the **LBT slot** time. An LBT slot is defined as a function of the worst case busy-sense time. In figure 4.1, suppose nodes A and C attempt to initiate a conversion with B at approximately the same time. If A determines that the channel is idle and begins transmitting an RFP frame at time 0, then the worst case busy-sense time is equal to the time, t, at which C begins sending a poll frame.

The backoff algorithm is executed, whenever a collision is suspected, to randomly distribute retries over an increasing number of LBT slots. The channel access algorithm used on radio channels treats "busy sense" as a collision and assumes that a collision may have occurred whenever a response is missed, since collision detection is not available.

The bridge layer is responsible for indicating to the MAC layer that a packet is being sent as a response to a multicast or broadcast message. If a packet is being sent in response to a multicast message, then the MAC layer waits for a random delay period before transmitting the response.

Broadcast and multicast frames are never retransmitted at the MAC layer.

The retry time of the MAC must be relatively short so that lost nodes can be detected quickly. When the MAC layer reports a failure to deliver a message to the bridging layer, the bridging layer can 1) save messages for SLEEPING terminals for later attempts, or 2) DETACH the node from the spanning tree.

The node identifier part of the MAC address is initially all 1's for all nodes except the root node. The all 1's broadcast address is used to by a node to send and receive address resolution packets, which are used to obtain a network address.

Address resolution.

Reverse Address Resolution Protocol (RARP) protocol overview.

A node which does not yet have a 16-bit network address must request a network address from a RARP server in the root node. The node uses an 11-bit node ID of all 1's until a unique ID is assigned by the RARP server. A RARP_request packet, containing the requesting node's unique 48-bit long ID and/or an alias (i.e. LPT1) for the 48-bit identifier, is sent to the server by the requesting node. The RARP server routes a RARP_response packet back to the requesting node using temporary RARP routing tables. A table entry is created in each node on the path to the root when the request is sent. When the RARP_response packet returns upstream, each node records the network address assigned to the requesting node. On the last hop, the RARP_response packet is broadcast to the requesting node. If the packet is not delivered successfully, a retry is initiated by the bridging entity in the requesting node after a BRIDGE_TIMEOUT period expires. The parent of the requesting node can service the retry without forwarding the retry RARP_request to the root. In general, if a bridge node receives a RARP_request and the long identifiers in the request matches an entry in its RARP routing table, and that entry has a valid 16-bit network address, then the bridge node can simply return a RARP_response, with the 16-bit address, without forwarding the request to the root node. Note that RARP routing table entries are aged and discarded before the address can become invalid.

A requesting node can specify that it does not have a 48-bit ID by not including it as a parameter or by including an ID of all 0's. An alias is required if there is no 48-bit ID.

A requesting node can specify that it does not have an alias by not including it as a parameter, or by including an alias with a length of 0.

The long identifiers in a RARP.request match temporary entries in a RARP routing table 1) if matching 48-bit IDs and aliases are present in both, 2) if both contain a matching alias and neither has a 48-bit ID, or 3) if both contain a matching 48-bit ID and neither has an alias.

Note that the RARP request function serves two purposes: 1) A 16-bit network address is obtained, and 2) The long ID and/or alias of the requesting node are registered with the address server.

Address Resolution Protocol (ARP) protocol overview.

A node can request the 16-bit network address of another node by sending an ARP.request packet to an ARP server in the root node. The ARP.request packet must contain either the 48-bit identifier or the alias of the target node. The ARP server returns the 16-bit network address of the target node in an ARP.response packet.

Network Management.

An SNMP-like network management provided is provided through a dedicated network control point which exists in a controller device. A network manager can retrieve and manipulate supported variables in devices distributed throughout the network to control devices and diagnose problems.

Boot Protocol.

A system load protocol similar to IEEE 802.1E is implemented by maintaining a load file database in a controller. Devices can request load files by class or by name. Alternatively, a network manager can distribute load files to devices over the network and can cause devices to reboot with the SNMP-like network management protocol described above.

References.

[1] B. M. Leiner, D. L. Nielson, and F. A. Tobagi, "Issues in Packet Radio Design", Proceedings of the IEEE, Vol. 75, No. 1, January 1987.

[2] L. Kleinrock and F. A. Tobagi, "Packet Switching in Radio Channels: Part I - Carrier Sense Multiple Access Modes and the Throughput Delay Characteristics", IEEE Transactions on Communications, Vol. COM-23, No. 12, December 1975.

[3] L. Kleinrock and F. A. Tobagi, "Packet Switching in Radio Channels: Part II - The Hidden Terminal Problem in Carrier Sense Multiple Access and the Busy Tone Solution", IEEE Transactions on Communications, Vol. COM-23, No. 12, December 1975.

[4] L. Kleinrock and F. A. Tobagi, "Packet Switching in Radio Channels: Part IV - Stability Considerations and Dynamic Control in Carrier Sense Multiple Access", IEEE Transactions on Communications, Vol. COM-25, No. 10, October 1977.

[5] L. Kleinrock and J. Silvester, "On the Behavior of Multi-hop Packet Radio Networks", Proceedings of the IEEE, Vol. 75, No. 1, January 1987.

[6] M. B. Pursley, "The Role of Spread Spectrum in Packet Radio Networks", Proceedings of the IEEE, Vol. 75, No. 1, January 1987.

[7] J. O. Onunga and R. W. Donaldson, "Performance Analysis of CSMA with Priority Acknowledgments (CSMA/PA) on Noisy Data Networks with Finite User Population", IEEE Transactions on Communications, Vol. 39, No. 7, July 1991.

[8] L. Kleinrock and J. Silvester, "Spatial Reuse in Multihop Packet Radio Networks", Proceedings of the IEEE, Vol. 75, No.1, January 1987.

[9] F. Backes, "Transparent Bridges for Interconnection of IEEE 802 LANs", IEEE Network, Vol. 2, No. 1, January 1988.

[10] International Standard ISO/DIS 8802-2.2.

[11] A. S. Tanenbaum, "Computer Networks", Prentice Hall, Second Edition

[12] D. E. Comer, "Internetworking with TCP/IP", Prentice Hall

[13] D. Cherition, "VMTP: Versatile Message Transaction Protocol, protocol specification", TCP/IP Network Working Group, RFC 1045

[14] IEEE standard 802.1E-1990, "Local and Metropolitan Area Networks, System Load Protocol"

[15] M. T. Rose, "The Simple Book"

Express Mail Label
No. EV 331533572US

Attorney Docket No.
14406US02

APPENDIX D

Title : Radio Frequency Local Area Network

Inventors: Meier, et al.

Attorney Docket No. 14406US02

SST NETWORK FRAME FORMATS

General format.

Pre-amble	Flag	MAC Header	Bridge Header	Bridge Data	LLC Header	LLC Data	CRC	Flag

General Field Definitions.

Preamble	1 to 8 bytes
Flag start delimiter	1 byte
MAC Destination Address	2 bytes
MAC Source Address	2 bytes
MAC Control	1 byte
Bridge Control	2 bytes
Bridge Destination Address	2 bytes
Bridge Source Address	2 bytes
Bridge Packet Parms	packet type dependent
Bridge Packet Optional Parms	M bytes
LLC DSAP	1 byte
LLC SSAP	1 byte
LLC Control	1 or 4 bytes
LLC Data	N bytes
CRC-CCITT	2 bytes
Flag end delimiter	1 byte
(optional trailer)	1 or 2bytes

16-bit Network Address Format.

bit 15	Mulitcast Flag
0	unicast frame
1	multicast or broadcast frame

bit 14-11	Node Type
0001	SS Terminal
0010	UHF Terminal
0011	All Terminals
0100	Bridge
1111	All Nodes

bit 10-0	Node Identifier
all 0's	root node identifer
all 1's	node without a network node identifier or any node

The well-known address of the root node is binary 0010 0000 0000 0000 (hexidecimal 2000).

MAC Control Byte (8 bits).

Bits 7-5 in the MAC control bytes are used to specify the frame type. MAC frames are one of two basic types: 1) request, or 2) poll, depending on the state of the R/P bit.

Request frame types.

000	EOD (end-of-data)
001	DATA
010	ENQ (enquiry)
011	RFP (request-for-poll)

Poll frame types.

100	WFP (wait for poll)
101	REJECT
110	CLEAR
111	POLL

Request control byte:

bit 7	R/P	0 = request frame
bit 6	DATA	0 = data frame
bit 5	MORE	1 = middle of bracket (DATA) 0 = end of bracket (EOD) 1 = RFP 0 = ENQ
bit 4	reserved	must be zero
bit 3	PRIORITY	0 = normal, 1 = high
bit 2	SEQ	sequence number, modulo 2
bit 1-0	LAN ID	00, 01, 10 or 11

Poll control byte:

bit 7	R/P	1 = poll frame
bit 6	WAIT	0 = wait for poll
bit 5	MORE	0 = clear
bit 4	reserved	must be zero
bit 3	reserved	must be zero
bit 2	SEQ	sequence number, modulo 2
bit 1-0	LAN ID	00, 01, 10 or 11

Bridge Control Bytes (16 bits).

bit 15-14	Bridge Header Format	00 = multihop, 01 = point-to-point
bit 13	(reserved)	must be zero
bit 12	REQ/RSP	0 = request, 1 = response
bit 11	(reserved)	must be zero
bit 10-8	Packet Type	(see table below)
bit 7	Bridge Parms	1 = optional bridge parms
bit 6	RSPRQ	1 = end-to-end response packet required
bit 5-3	Protocol	000 = no data-link data, 001 = LLC data
bit 2	ATTI	1=attach indication
bit 1-0	(reserved)	must be zero

Bridge Packet Types.

000	Data Packet
001	(reserved)
010	Hello Packet
011	Attach Packet
100	Detach Packet
101	Address Resolution Packet
110	Reverse Address Resolution Packet
111	(reserved)

Optional Bridge Parameters - general format.

1-bit end-of-parms flag	1 = last optional parm
7-bit parm type	(see table below)
1-byte parm length	length of parm value field in bytes
M-byte parm value	(value or list of values)

Optional Parameters.

Parm Type	Parm Length	Description
01h	2 bytes	A 2-byte network address.
02h	6 bytes	Long Identifier.
03h	M*2	Decendant List. A list of 2-byte addresses.
04h	N*2	Detached List. A list of 2-byte addresses.
05h	P*2	Pending Message List. A list of 2-byte addresses.
06h	2 bytes	Distance (cost) from the root.
07h	Q bytes	Well-known alias.
08h	R bytes	Forward List. A list of 2-byte addresses.
09h	1 byte	Load Indicator. An indication of the channel load based on frame frequency.
0Ah	S bytes	Well-known alias of the root.
0Bh	6 bytes	Long identifier of the root.
0Ch	1 or 2 bytes	Awake time (in 100 millisecond units). All 1's denotes forever.
0Dh	1 or 2 bytes	Awake time offset (in 100 millisecond units).
0Eh	1 byte	Delivery service type. 0=deliver immediately. 1=store until the node is awake. 2=attempt to deliver immediately, then store until the node is awake.
0Fh	1 byte	Maximum stored message count. The maximum number of hello times that the parent node should store a message for the source child node.
10h	2 bytes	Decendent count.

Bridge Request Packet Formats.

Data (Type 000).

MAC Header	
Bridging Header	
Optional Bridging Parms - Max. stored message count. - Delivery service type. - Wake up time. - Wake up time offset.	N bytes
LLC Header (optional)	
LLC Data (optional)	

Hello (Type 010).

MAC Header
Bridging Header

Attach (Type 011).

MAC Header	
Bridging Header	
Optional Bridging Parms - Decendant list - Max. stored message count. - Delivery service type. - Wake up time. - Wake up time offset.	N bytes
LLC Header (optional)	
LLC Data (optional)	

Detach (Type 100).

MAC Header	
Bridging Header	
Optional Detach Parms - Decendant list - Forward list	N bytes

Address Resolution (Type 101).*

MAC Header	
Bridging Header	
ARP Operation	1 byte
bit 7-4 (reserved)	must be zero
bit 3-0 reason code	0 = ok, other = error code
Network Address	2 bytes
Long ID/Alias type	1 byte
Long ID/Alias length	1 byte
Long ID/Alias	N bytes

*The Long ID/Alias can be a 6-byte identifier or an Alias. The address server will set the network address field to the network address of the associated node. If the Long ID (or Alias) cannot be found the network address field will be set to all 1's.

Reverse Address Resolution (Type 110).*

MAC Header	
Bridging Header	
RARP Operation	1 byte
bit 7 New Alias	1 = replace existing Alias
bit 6 New Long ID	1 = replace existing Long ID
bit 5 (reserved)	must be zero
bit 3-0 reason code	0 = ok, other = error code
Network address	2 bytes
Long ID type (02h)	1 byte
Long ID length (6)	1 byte
Long ID	N bytes
Alias type (07h)	1 byte
Alias length	1 byte
Alias	N bytes

*The requesting node must set the Long ID field and/or the Alias field. The source bridge address must be set to the source node type and a node ID of all 1's. The address server will set the network address field to the next available 16-bit address. If an address is not available, the field will be set to all 1's.

Bridge Response Packet Formats.

Hello (Type 010).

MAC Header	
Bridging Header	
Cost-to-root	2 bytes (0xFFFF = infinity)
Seed/Attach priority bit 7-2 seed bit 1-0 attach priority	1 byte 00 = lowest priority
Offset	1 byte 0-254 = transmission offset time in hundredths of seconds. 255 = unscheduled.
Root Priority bit 7-6 reserved bit 5-3 user priority bit 2-0 device priority	1 byte (must be zero) 000 = lowest priorty. 000 = lowest priority.
Root ID Sequence Number	1 byte
Optional fields - Root ID - Pending Message List - Decendant Count - Detached List - Load Indicator	N bytes

Attach (Type 011).

MAC Header
Bridging Header

Detach (Type 100).

MAC Header
Bridging Header

Address Resolution (Type 101).*

MAC Header	
Bridging Header	
ARP Operation	1 byte
bit 7-4 (reserved)	must be zero
bit 3-0 reason code	0 = ok, other = error code
Network Address	2 bytes
Long ID/Alias type	1 byte
Long ID/Alias length	1 byte
Long ID/Alias	N bytes

*The Long ID/Alias can be a 6-byte identifier or an Alias. The address server will set the network address field to the network address of the associated node. If the Long ID (or Alias) cannot be found the network address field will be set to all 1's.

Reverse Address Resolution (Type 110).*

MAC Header	
Bridging Header	
RARP Operation	1 byte
bit 7 New Alias	1 = replace existing Alias
bit 6 New Long ID	1 = replace existing Long ID
bit 5 (reserved)	must be zero
bit 3-0 reason code	0 = ok, other = error code
Network address	2 bytes
Long ID type (02h)	1 byte
Long ID length (6)	1 byte
Long ID	6 bytes
Alias type (07h)	1 byte
Alias length (N)	1 byte
Alias	N bytes

*The requesting node must set the Long ID field and/or the Alias field. The source bridge address must be set to the source node type and a node ID of all 1's. The address server will set the network address field to the next available 16-bit address. If an address is not available, the field will be set to all 1's.

Express Mail Label
No. EV 331533572US

Attorney Docket No.
14406US02

APPENDIX E

Title : Radio Frequency Local Area Network

Inventors: Meier, et al.

Attorney Docket No. 14406US02

BRIDGING LAYER SPECIFICATION

Introduction.

This document, in conjunction with the SST network frame format specification, defines the routing layer protocol used with the SST network.

Functional Requirements.

The bridging layer:

- selects a root bridge node.

- organize nodes in the network into an optimal spanning tree rooted at the root bridge.

- routes terminal-to-terminal data traffic along branches of the spanning tree. Note that a host computer is treated as a terminal. A routing table is maintained in each bridge node for all upstream nodes. The path to downstream nodes is inherent in the structure of the spanning tree.

- provides a service for storing packets for SLEEPING terminals.

- propagates lost node information throughout the spanning tree.

- maintains the spanning tree links.

- distributes network interface addresses.

- organizes nodes into logical coverage areas on radio channels.

Bridg Packet Definitions.

Bridge Control Byte Fields.

Bridge header format - This field is used to define the format of the bridge header.

00 = multihop. This is the normal bridge header format. The bridge header includes source and destination address fields.

01 = point-to-point. The bridge header does NOT include source and destination address fields.

Packet type - This field is used to specify the bridging layer packet type.

Bridge parms - If this bit field is set ON then optional bridging layer parameters immediately follow the bridge header.

RSPRQ - This field can be used to request an end-to-end bridging layer response packet. Normally this field should be set ON for ATTACH, RARP and ARP request packets, and should be set OFF for all other bridging layer packets.

ATTI - A bridge node will set this bit ON in an ATTACH.request packet whenever the source node is not in the bridge's routing table. The bit value in an ATTACH.response packet follows the state of the bit in the associated ATTACH.request packet received by the root node. If a terminal receives an ATTACH.response packet with the ATTI bit set ON, it is a positive indication that the terminal was detached and has reattached to the network.

Protocol - This field is used to indicate the presence and type of higher layer data.

000 = no higher layer data is contained in the packet.
001 = an LLC message is contained in the packet.
010 = an NNMP message is contained in the packet.

Bridge Packet Types.

DATA.request.

DATA.request packets are general purpose bridging layer packets used to send higher layer data and/or bridging layer parameters.

A child node can periodically send a (possibly empty) DATA.request packet to its parent to ensure that it is still attached to the spanning tree. Note that an ATTACH.request packet, with the ATTI bit set ON, is generated by a bridge node whenever the bridge node receives a downstream DATA.request packet and the source of the DATA packet is not in the bridge's routing table.

HELLO.request.

A HELLO.request packet is used to solicit unscheduled HELLO.response packets from bridge nodes. A HELLO.request packet can be broadcast by an unattached node when it wants to quickly reattach to the network.

HELLO.response.

HELLO.response packets are sent periodically at calculated time intervals by all bridge nodes. In addition, bridge nodes will broadcast a HELLO.response packet whenever a HELLO.request packet is received. HELLO.response packets are used to build the spanning tree and are used to advertise pending message information and lost node information.

ATTACH.request.

A node transmits an ATTACH.request packet, with the RSPRQ bit set ON, to attach to the network. In addition, a node must transmit an ATTACH.request packet at least once per ATTACH_TIMEOUT time period to maintain its path in the network. All ATTACH.request packets are implicitly forwarded to the root bridge node. If the RSPRQ bit is set ON the root bridge node will return an ATTACH.response packet.

Higher layer data can be piggybacked onto ATTACH.request packets by setting the bridging layer destination address to the 16-bit address of the node for which the data is intended. If data is piggybacked onto an ATTACH.request packet, the bridging layer will split the ATTACH packet into separate ATTACH and DATA request packets as soon as the next hop to the destination address is not on the path to the root node (i.e. the first upstream hop). The destination address of the generated ATTACH.request packet is the well-known address of the root node.

If a bridge node receives a downstream DATA.request packet and the source node is not in the bridge's routing table, then the bridge node will automatically generate an ATTACH.request packet, with the ATTI bit set ON, for the source node and forward it to the root node. The source address in the generated ATTACH.request packet is the same as the source address in the DATA packet. (Note that the DATA packet can simply be converted to an ATTACH packet.)

ATTACH.response.

The root node will return an ATTACH.response packet to the source node in the associated ATTACH.request packet if the RSPRQ bit in the request packet is set ON. The ATTI bit will be set ON if the originating node was not fully connected to the network (i.e. if the node was not in the routing table of a bridge node on the path of the request packet).

Detach.request.

DETACH.request packets are used to notify the network that a node has detached. DETACH.request packets can be reliably broadcast throughout the spanning tree by using the reliable flooding mechanism described below. If a DETACH.request packets includes a decedent list, then all nodes in the decedent list and the bridging layer source node are purged from the routing table of nodes which receive the request.

Address Resolution Packet (ARP).

An address resolution packet is used to acquire the 16-bit network address of a destination node, when only the alias or 48-bit identifier of the node is known. 16-bit network addresses are cached by the bridging layer. An ARP packet is generated automatically by the bridging layer whenever it receives a send request for which the destination is not in its cache.

Reverse Address Resolution Packet (RARP).

A RARP packet is used to set or change the alias and/or 48-bit long identifier of a device and to acquire a 16-bit network address.

Constants, timers and data structures.

ROOT_ADDRESS = hex 2000;

HELLO_RETRY = 5;
MAX_HELLO_LOST = 8;
HELLO_SLOT_SIZE = .020 seconds.
HELLO_MOD_VAL = 67.
AVG_HELLO_SLOTS = 100.
MIN_HELLO_SLOTS = HELLO_MOD_VAL.
MAX_HELLO_SLOTS = (HELLO_MOD_VAL * 2) - 1.
AVG_HELLO_PERIOD = HELLO_SLOT_SIZE * AVG_HELLO_SLOTS seconds;
MIN_HELLO_PERIOD = HELLO_SLOT_SIZE * MIN_HELLO_SLOTS seconds;
MAX_HELLO_PERIOD = HELLO_SLOT_SIZE * MAX_HELLO_SLOTS seconds;
HELLO_TIMEOUT = HELLO_RETRY * AVG_HELLO_PERIOD + 1 seconds;
S_HELLO_WAIT = .5 seconds;
MAX_HELLO_OFFSET = 200 milliseconds;

BRG_RSP_TIMEOUT = 10 seconds;

HOLD_DOWN_TIME = MAX_HELLO_PERIOD*(MAX_HELLO_LOST+2) seconds;
DEF_SAVE_CNT = 3;
MAX_SAVE_CNT = 5;
AWAKE_TIME_UNIT = .1 seconds.

ADDRESS_TIMEOUT = 30 minutes.
MAX_ADDRESS_LIFE = 120 minutes.

NETWORK_TIMEOUT = 12 minutes.

ROUTE_TIMEOUT = 10 minutes.
ATTACH_TIMEOUT = 8 minutes.

UT_OOR_TIME = 20 seconds; (Unattached terminal out-of-range list entry timeout)
UB_OOR_TIME = 100 seconds; (Unattached bridge out-of-range list entry timeout)
AT_OOR_TIME = 100 seconds; (Attached terminal out-of-range list entry timeout)
AB_OOR_TIME = 300 seconds; (Attached bridge out-of-range list entry timeout)

UT_OOR_AGED_TIME = 300 seconds; (Attached terminal out-of-range list aged entry timeout)
UB_OOR_AGED_TIME = 500 seconds; (Attached bridge out-of-range list aged entry timeout)
AT_OOR_AGED_TIME = 500 seconds; (Attached terminal out-of-range list aged entry timeout)
AB_OOR_AGED_TIME = 1000 seconds; (Attached bridge out-of-range list aged entry timeout)
B_ENABLE_WAIT = 20 seconds;
R_IDLE_TIME = 60 seconds;
MAX_RARP_RETRY = 10;
MAX_ATTACH_RETRY = 5;
B_TIMEOUT; (Bridging layer timeout value - a function of the cost to the root.)

HOP_COST_ETHER = 20;
HOP_COST_485LAN = 40;
HOP_COST_SSRADIO = 125;
HOP_COST_BUS = 0;

OOR_COST = 75;

A **hello-timer** is used to measure a HELLO learning period.

An **inactivity-timer** is used to measure periods of network inactivity.

A **path-timer** is used to measure the maximum round-trip path delay for bridging layer packets which require a response.

In any enabled state, all nodes, except the root, maintain two lists: an **in-range list** and an **out-of-range list.**

An entry in the in-range list contains the network address of a bridge node which was the source of a HELLO packet, the port used to communicate with the bridge node, an aging factor, and path cost information. Each entry in the in-range list is aged so that it is discarded if no HELLO packet is received from the associated bridge within HELLO_TIMEOUT seconds. The aging factor for an entry in the in-range list is reset to zero whenever a HELLO packet is received. The aging factor for the entry of the parent node is reset to zero whenever any packet is received from the parent. In-range entries are classified as OLD after two HELLO periods pass without receiving a HELLO packet from the bridge specified in the entry.

An entry in the out-of-range list contains the network address of a bridge node, the port used to communicate with the bridge, and an aging factor. The out-of-range list is used to remember failed paths when a node is attempting to attach to the network. Entries in the list are aged to indicate how long entries have been in the list. Out-of-range entries are classified as OLD after two HELLO periods in the list. Out-of-range entries are classified as AGED after OOR_TIME seconds in the list. AGED entries which are also in the in-range list are not selected as a potential parent node unless no other in-range entries are available. AGED entries are discarded after OOR_AGED_TIME seconds have passed.

Each node maintains a **current_root** variable, which is initialized to NULL. The variable contains the priority, root sequence, and ROOT ID the currently active root node, where ROOT ID is a 48-bit device ID and/or alias. The variable is used to remember the current root

node. Attached nodes detach and go into an initial unattached state, without a network address, when the root node changes.

In general, an entry is added/updated in the in-range list of an enabled, attached or unattached node whenever a HELLO packet, with the priority, sequence, and identifier of the current root node, is received, and the source bridge is not in the node's out-of-range list.

Each node, except the root, maintains a **current_parent** variable, which contains the network address and path cost information of the current parent of the node when the node is in an attached state.

MSG_PENDING is (conceptually) a global state variable which is true when an incoming message is expected.

State Transition Logic - notation and assumptions.

Assumptions.

The state transition logic in this document and the SST network assumes the following:

- All nodes are assigned a LAN ID.
- Packets which do not belong to the network specified by a node's LAN ID are discarded by the MAC layer and are not passed to the bridging layer.
- In any enabled state, all nodes store information derived from received HELLO.response packets.
- HELLO.response packets with a cost-to-root field value of 0 are from the root node.
- All nodes remain awake in any enabled unattached state. If SLEEPING nodes are unable to attach to the network after a maximum awake time has expired, an error is returned to a higher layer, and retries are handled by the higher layer.
- Root candidates must have a direct-link host port.
- All root candidates should be wired together.
- A node loses its address and goes into an intermediate hold-down state whenever a new root node is detected.

State Notation.

State names used in this document are hierarchical. Suffixes are added to state identifiers to further refine a state definition. An unqualified high-order prefix may be used to reference all possible substates.

The bridging entity in each node is in one of the following high-level **node states**.

R - Root node. The node owns the root node address.
RC - Root candidate node. The node does not have an address.
BB - Bridge node with a root priority of zero.
BR - Bridge node which has a non-zero root priority.
B - Any Bridge node.
T - Terminal node.
* - any node.

a - subscript used to qualify a node state to indicate that the node does not have a network address. For example, T_a is used to specify a terminal node that does not have a network address.

All node states are further qualified by one of three **attach states**:

D - The node is Disabled and unattached.
U - The node is enabled and Unattached.
A - The node is enabled and Attached to the network.

A node can be in an intermediate hold-down state:

I - The node is in an Intermediate hold-down state. Bridge nodes, which were attached, transmit HELLO.response packets with an infinite cost in the intermediate state.

As an example, RC.U, is used to denote the node and attach state of a root candidate which is not attached to an SST network.

The following substates are used to qualify an unattached node:

idle - No network activity has been detected.
wait - Wait for the first HELLO.response packet.
hello - A HELLO.response packet has been received.
rarp - An address request is in progress.
attach - An attach request is in progress.

Root Resolution Protocol.

Each SST network must have one or more root candidates. Each root candidate node enters a RC.U state when the bridging entity in the node is enabled. This state ends when 1) the root candidate determines that a higher priority root node already exists and enters the BR.U state, or 2) the root candidate assumes ownership of the root node status and enters the R.A state. A node in any BR state assumes the root node status if 1) the network becomes idle, or 2) a lower priority root node is detected.

A root candidate which does not detect any activity assumes the root node status. If activity is detected, the root candidate remains in the RC.U.wait state until a HELLO.response packet is received or until network activity ceases.

When an unattached non-candidate node receives its first HELLO.response packet it enters the *.U.hello state, and sets a hello-timer used to measure a HELLO learning period. The hello-timer expires after HELLO_RETRY HELLO periods.

In the R.A state the root node broadcasts a HELLO.response packet once per HELLO_PERIOD time period, according to a random distribution algorithm. The root HELLO.response packets contain a path cost of 0, the priority of the root node, a root ID sequence number, and a ROOT ID which is either the unique long identifier or the unique alias of the root device. **The priority, ROOT ID sequence, and ROOT ID fields are copied into the HELLO.response packets transmitted by all non-root bridges in the network.**

A ROOT ID sequence number is stored in non-volatile storage by all root candidates. The sequence number is copied into RAM by the root node when it determines that it is the root and the copy in non-volatile storage is incremented. The copy in RAM is included in all HELLO.response packets broadcast by all attached bridge nodes in the spanning tree.

Hello packet priority.

A "higher priority HELLO.response packet" is defined as any HELLO.response packet which contains a matching LAN ID and either 1) a higher concatenated USER PRIORITY+DEVICE PRIORITY field, or 2) an equal priority field and a higher priority ROOT ID. A ROOT ID can consist of a unique 48-bit device ID and/or a device alias. A "higher priority ROOT ID" is defined as 1) the ID with the higher 48-bit ID, or, 2) if neither candidate has a 48-bit ID, the ID with the alias with highest string value. Note that if the ROOT ID does not contain a unique 48-bit device ID, then the 48-bit device ID is assumed to be all 0's.

It may be possible for a root candidate to receive a HELLO.response packet with an equal priority if the ROOT ID field in the HELLO.response packet matches the candidate's device identifier. HELLO.response packets with a ROOT ID field that matches the identifier of the local device and a non-zero path cost are assumed to be associated with an out-of-date spanning tree and are discarded by the bridging layer. A matching ROOT ID and a zero path cost causes a fatal error.

Root resolution state transition table.

The state transition table below defines transitions in the root resolution process.

state	event	action	next state
RC.D	Bridging entity enabled.	Enable MAC on all network ports; set HELLO_TIMEOUT inactivity-timer.	RC.U.idle
RC.U.idle	Inactivity-timer expires.		R.A
	Non-HELLO packet received.	Set R_IDLE_TIME inactivity-timer.	RC.U.wait
	Higher priority HELLO packet received.	Set R_IDLE_TIME inactivity-timer; set HELLO_TIMEOUT hello-timer.	BRa.U.hello
	Lower priority HELLO packet received.		R.A
RC.U.wait	Inactivity-timer expires.		R.A
	Non-HELLO packet received.	Set R_IDLE_TIME inactivity-timer.	RC.U.wait
	Higher priority HELLO packet received.	Set R_IDLE_TIME inactivity-timer; set HELLO_TIMEOUT hello-timer.	BRa.U.hello
	Lower priority HELLO packet received.		R.A
BR.U	Lower priority HELLO packet received.	Set R_IDLE_TIME inactivity-timer.	BR.I then RC.U.wait
R.A	Higher priority HELLO packet received.	Transmit HELLO packets with an infinite path cost for MAX_HELLO_LOST+1 HELLO periods; set R_IDLE_TIME inactivity-timer; set HELLO_TIMEOUT hello-timer.	R.I then BRa.U.hello

Attaching to the Spanning Tree.

Nodes learn the best path to the root node by listening to HELLO.response packets from attached bridge nodes. After a learning time expires, a node without a network address sends a RARP.request packet to the attached bridge, in the in-range list, which provides the best path to the root. A node with a network address sends an ATTACH.request packet to the attached bridge in the in-range list which provides the best path. The "best path" is primarily a function of the path cost to the root.

Attach Selection Criteria.

The criteria for selecting a parent bridge node as the first hop in a best path is defined as follows: 1) first, only nodes which are in the in-range list, and not in the out-of-range list, are considered; 2) if no such node exists, then nodes which are in the in-range list but are classified as AGED, in the out-of-range list, are considered; 3) the node with the lowest cost to the root is chosen from the set of "parent candidates"; 4) if two or more nodes have the same path cost, then the node with the best signal strength is chosen (if a signal strength indicator is available); 5) if both the path cost and signal strength are equal, then the node with the highest attach priority is chosen; 6) if the path cost, signal strength and attach priority are equal, then the node with the lowest network address is chosen. Elements 5 and 6 of the selection criteria are intended to group terminals into logical coverage areas under the control of a single bridge node, rather than allowing terminals to randomly attach to any bridge node when physical coverage areas overlap.

All parent candidates must be in the in the in-range list. New entries in the out-of-range list are not considered as parent candidates. If no parent candidates exist an unattached node can wait and listen, or, optionally, can solicit short HELLO.response packets by transmitting a global HELLO.request packet.

Bridge Attach State Transitions.

The state transition table below defines the state transitions required for a disabled bridge node to obtain a network address. Note that a root candidate enters the B_n.U.hello state after learning of a higher priority root.

state	event	action	next state
BB.D	Bridging layer is enabled.	Enable MAC on all network ports; initialize in-range list to NULL; initialize out-of-range list to NULL.	B_n.U.wait
B_n.U.wait	HELLO packet received from the root.	Send RARP.request to the root; set B_TIMEOUT path-timer; set retry count to 1.	B_n.U.rarp
	HELLO packet received from a non-root bridge.	Set hello-timer.	B_n.U.hello
B_n.U.hello	Hello-timer expires or HELLO packet received from the root node.	Send RARP.request to the attached bridge with the best path; set B_TIMEOUT path-timer; set retry count to 1.	B_n.U.rarp
B_n.U.rarp	RARP.response packet received.	If address was received send ATTACH.request to the attached bridge with the best path; set B_TIMEOUT timer; set retry count to 1.	If address was received then goto B.U.attach else post error; goto BB.D or RC.D.
	path-timer expires.	Resend RARP.request. Increment retry count.	B_n.U.rarp
	Retry count > MAX_RARP_RETRY.	Initialize in-range list to NULL.	B_n.U.wait
	MAC layer retry error.	Add respective bridge to out-of-range list. Delete bridge from in-range list. If list is not empty, send a RARP.request to the attached bridge with the best path; set B_TIMEOUT timer; set retry count to 1.	If in-range list is empty then goto B_n.U.wait else goto B_n.U.rarp

The transition table below defines the state transitions required for an unattached bridge node, with a network address, to attach to the network. Note that an attached bridge node enters the B.U.wait state after detaching from the network. An unattached bridge node enters the B.U.attach state after receiving its network address.

state	event	action	next state
B.U.wait	HELLO packet received from the root.	Send ATTACH.request to the root; reset B_TIMEOUT timer; set retry count to 1.	B.U.attach
	HELLO packet received from a non-root bridge.	Reset hello-timer.	B.U.hello
B.U.hello	Hello-timer expires or HELLO packet received from the root node.	Send ATTACH.request to the attached bridge with the best path; reset B_TIMEOUT timer; set retry count to 1.	B.U.attach
B.U.attach	ATTACH.response packet received from bridge with best path in in-range list.		B.A
	ATTACH.response packet received from bridge other than one with best path.	(ignore)	B.U.attach
	B_TIMEOUT timer expires.	Resend ATTACH.request. Increment retry count.	B.U.attach
	Retry count > MAX_ATTACH_RETRY.	Initialize in-range list to NULL.	B.U.wait
	MAC layer retry error.	Delete bridge from in-range list. If list is not empty, send an ATTACH.request to the attached bridge with the best path; set B_TIMEOUT timer; set retry count to 1.	If in-range list is empty then goto B.U.wait else goto B.U.attach
	HELLO packet received with shorter path cost and not in out-of-range list.	Send ATTACH.request to the attached bridge with the best path; reset B_TIMEOUT timer; set retry count to 1.	B.U.attach

Note that the ATTACH.request packets, specified in the above transition tables, require an end-to-end response (i.e. an ATTACH.response packet).

State transition tables for an unattached bridge node which has a non-zero priority (BR.U) are identical to the above state tables for bridges with a priority of zero, with the following exceptions: 1) An unattached bridge with a non-zero priority assumes the root node status whenever no packets are received within an R_IDLE_TIME inactivity period in any state; 2) an unattached bridge with a non-zero priority enters the RC.U.wait state, after a BR.I hold-down period, whenever a lower-priority HELLO message is received. Note that an inactivity-timer is constantly running and must be reset whenever a packet is received.

In any attached or unattached state, a bridge loses its address, waits for a hold-down period, and then goes to a *.U.wait state, whenever a new root node is detected.

Terminal Attach State Transitions.

The state transition table below specifies the state transitions required for a disabled terminal node to attach to the network.

state	event	action	next state
T.D	Bridging layer is enabled.	Enable MAC on all network ports; initialize in-range list to NULL; initialize out-of-range list to NULL; set HELLO_TIMEOUT hello-timer.	T_a.U.wait
T_a.U.wait	HELLO packet received from the root.	Send RARP.request to the root; set B_TIMEOUT timer; set retry count to 1.	T_a.U.rarp
	HELLO packet received from a non-root bridge.	Reset HELLO_TIMEOUT hello-timer.	T_a.U.hello
	Hello-timer expires.	(sleep); reset HELLO_TIMEOUT hello-timer.	T_a.U.wait
T_a.U.hello	Hello-timer expires or HELLO packet received from the root node.	Send RARP.request to the attached bridge with the best path; set B_TIMEOUT timer; set retry count to 1.	T_a.U.rarp
T_a.U.rarp	RARP.response packet received.	If address was received send ATTACH.request to the attached bridge with the best path; set B_TIMEOUT timer; set retry count to 1.	If address was received then goto T.U.attach else post error; goto T.D
	B_TIMEOUT timer expires.	Resend RARP.request. Increment retry count.	T_a.U.rarp
	Retry count > MAX_RARP_RETRY.	Initialize in-range list to NULL.	Ta.U.wait
	MAC retry error.	Add respective bridge to out-of-range list. Delete bridge from in-range list. If list is not empty, send a RARP.request to the attached bridge with the best path; set B_TIMEOUT timer; set retry count to 1.	If in-range list is empty then goto T_a.U.wait else goto T_a.U.rarp

In any attached or unattached state, a terminal loses its address, waits for a hold-down period, and then goes to the T_a.U.wait state, whenever a new root node is detected.

The transition table below defines the state transitions required for an unattached terminal node, with a network address, to attach to the network. Note that an attached terminal node enters the T.U.wait state after detaching from the network. If the terminal has a message pending, then the terminal will solicit short HELLO.response packets with a global HELLO.request and will wait HELLO_WAIT seconds for a HELLO packet.

state	event	action	next state
T.U.wait	HELLO packet received from the root.	Send ATTACH.request to the root; set B_TIMEOUT timer; set retry count to 1.	T.U.attach
	HELLO packet received from a non-root bridge.		T.U.hello
	Hello-timer expires.	Pause (and sleep); send global HELLO.request; increment retry count ; Reset S_HELLO_WAIT hello-timer.	T.U.wait
	Hello-timer expires; retry count > MAX_HELLO_WAIT.	Return uncompleted requests with error.	Ta.U.wait
T.U.hello	Hello-timer expires or HELLO packet received from the root node.	Send ATTACH.request to the attached bridge with the best path; set B_TIMEOUT timer; set retry count to 1.	T.U.attach
T.U.attach	ATTACH.response packet received from bridge with best path in in-range list.	Set current_parent to best path bridge.	T.A
	ATTACH.response packet received from bridge other than one with best path (i.e. in response to an old request).	(ignore)	T.U.attach
	B_TIMEOUT timer expires.	Resend ATTACH.request; increment retry count.	T.U.attach
	Retry count > MAX_ATTACH_RETRY.	Initialize in-range list to NULL; send global HELLO.request; set retry count to 1; Set S_HELLO_WAIT hello-timer.	T.U.wait
	MAC layer retry error.	Move bridge from in-range list to out-of-range list. If in-range list is empty, send a global HELLO.request; set S_HELLO_WAIT hello-timer; set retry count to 0; else, if in-range list is not empty, send an ATTACH.request to the attached bridge with the best path; set B_TIMEOUT path-timer; set retry count to 1.	If in-range list is empty then T.U.wait else T.U.attach

	Better path found.	Send ATTACH.request to the attached bridge with the best path: set B_TIMEOUT path-timer; set retry count to 1.	T.U.attach

In any attached or unattached state, a terminal loses its address, waits for a hold-down period, and then goes to the T_a.U.wait state, whenever a new root node is detected.

Reliable Flooding Mechanism.

Information can be flooded throughout the network by using a "reliable flooding mechanism". A node can flood a bridging layer request packet 1) by setting the MAC layer and bridging layer destination addresses to all 1's (i.e. a global address), 2) by setting the RSPRQ bit ON, and 3) by including a **forward list** in the packet. The forward list is identified by the bridging layer parameter type hexadecimal 08. It is used to specify which nodes must forward and acknowledge the request. Initially, the forward list consists of all bridge nodes which are either children or the parent of the node which generated the packet. Nodes in the forward list acknowledge the request packet with a unicast response packet of the same packet type. Note that only one flooded packet of a given type may be outstanding at a time. If a receiving node in the forward list is attached to one or more branches of the spanning tree, other than the branch on which the request packet arrived, then the node must rebroadcast the request packet. The forward list in the rebroadcast packet will consist of each bridge node which is the first hop on such other branches. Note that the forward list may be empty if the first (and only) hop in all other branches is a terminal node.

Spanning Tree Link Maintenance and Recovery.

A link in the spanning tree is lost whenever one of the following events occurs:

1) A child node is unable to deliver a message to its parent bridge node.
2) A child node finds a better path to the root node.
3) MAX_HELLO_LOST consecutive HELLO.response packets from a parent bridge node are lost by a child node.
4) A HELLO.response packet, from a parent bridge node or current root node, with an infinite path cost, is received by a child node.
5) A HELLO.response packet is received with a new ROOT ID or Root Sequence Number.
6) An attached node, which is not in a "DETACH HOLD_DOWN" state, receives a DETACH packet or HELLO.response packet with its address in the packet's detached node list.
7) A parent node is unable to deliver a message to a child node.
8) A routing table entry is aged and deleted.

- Response actions to events 1-6 are defined in the detach state transition tables below.
- Event 7 is discussed in the section which describes detach packet logic.
- Event 8 is discussed in the section on routing table maintenance.

Detach State Transitions.

Terminal Detach State Transitions.

The transition table below defines events which may cause a terminal to detach from the network.

state	event	action	next state
T.A	MAC_send error.	Remove old entries and the destination bridge from the in-range list; initialize the out-of-range list to the MAC destination bridge; If the in_range list is empty, send global HELLO.request; set S_HELLO_WAIT hello-timer; set retry count to 1; else, if in-range list is not empty, send an ATTACH.request to the attached bridge with the best path; set B_TIMEOUT path-timer; set retry count to 1.	If in-range list is empty then T.U.wait else T.U.attach
	Terminal address seen in a detached node list in a DETACH packet or HELLO packet and current time is past HOLD_DOWN.	Remove old entries from the in-range list; add the source of the DETACH or HELLO packet to the in-range list; initialize the out-of-range list to NULL; send an ATTACH.request to the attached bridge with the best path; set B_TIMEOUT path-timer; set retry count to 1; set HOLD_DOWN to current time plus HOLD_DOWN_TIME.	T.U.attach
	Better path found to the root and no data transactions are pending.	Issue a MAC_enquiry to the best path bridge.	T.A
	Positive response to a MAC_enquiry received from the best path bridge.	Send an ATTACH.request to the destination bridge.	T.U.attach
	Negative response to a MAC_enquiry.	Move the destination bridge from th in-range list to the out-of-range list.	T.A

	MAX_HELLO_LOST consecutive HELLO packets from the parent node are missed. (Note that sleeping terminals must count skipped HELLO times as misses.)	Send an ATTACH.request to the attached bridge with the best path: set B_TIMEOUT path-timer: set retry count to 1.	T.U.attach
	HELLO packet received from parent with an infinite path cost (same ROOT ID).	Remove parent and old entries from the in-range list: remove old entries from the out-of-range list: set HELLO_TIMEOUT HELLO timer.	T.U.wait
	HELLO packet received with new ROOT ID or Root Sequence Number.	Update current-root. Initialize node identifier to all 1's. Initialize in-range and out-of-range lists to NULL.	T_A.U.wait

A sleeping terminal should remain awake, whenever a threshold number (i.e. 1 or 2) consecutive HELLO packets have been missed from its parent node, until a HELLO packet is received from its parent or until the terminal detaches and reattaches to a different parent node.

A terminal which is waiting to receive a response time critical message can periodically transmit an empty DATA.request packet to its parent bridge node, to determine if the parent is still in range, if the message is not received in the expected amount of time.

A terminal which is in a DETACH HOLD_DOWN state must send an ATTACH.request packet to the root node after the HOLD_DOWN state ends, to ensure that it is fully attached to the network.

Bridge Detach State Transitions.

The transition table below defines events which may cause a bridge node to detach from the network. The "detach" action, specified in the table, consists of broadcasting infinite cost HELLO packets for MAX_HELLO_LOST+1 HELLO periods, in an intermediate detach (B.I.detach) state before going to an unattached state.

state	event	action	next state
B.A	MAC_send error.	Detach. Remove the MAC destination and old entries from the in-range list; initialize the out-of-range list to the MAC destination bridge; set HELLO_TIMEOUT hello-timer.	B.U.wait
	Bridge address seen in a detached node list in a DETACH packet or HELLO packet.	Detach. Remove old entries from the in-range list; add the source of the DETACH or HELLO packet to the in-range list; initialize the out-of-range list to NULL; set HOLD_DOWN_TIME hello-timer.	B.U.wait
	Better path found to the root.	Issue a MAC_enquiry to the best path bridge.	B.A
	Positive response to a MAC_enquiry received from the best path bridge.	Send an ATTACH.request with a decedent list to the destination bridge.	B.U.attach
	Negative response to a MAC_enquiry.	Move the destination bridge from the in-range list to the out-of-range list.	B.A
	MAX_HELLO_LOST consecutive HELLO packets from the parent node are missed.	Detach. Remove old entries from the in-range list; initialize the out-of-range list to the parent node; set MAX_HELLO_PERIOD hello-timer.	B.U.wait
	HELLO packet received from parent with an infinite path cost.	Detach. Remove parent and old entries from the in-range list; remove old entries from the out-of-range list; set HELLO_TIMEOUT HELLO timer.	B.U.wait
	HELLO packet received with new ROOT ID or Root Sequence Number and either bridge can not be a root candidate or packet has a higher priority.	Detach. Update current-root. Initialize node identifier to all 1's. Initialize in-range and out-of-range lists to NULL.	B_a.U.wait
BR.A	Lower priority HELLO packet received.	Detach. Initialize in-range and out-of-range lists to NULL.	R.A

Note that a bridge node will never be in an attached HOLD_DOWN state.

Detach packet logic.

A parent node generates a DETACH.request packet whenever it is unable to deliver a message to a child node. The two possible cases are 1) the child is a bridge, or 2) the child is a terminal.

Case 1 - The lost node is a bridge.

When a parent bridge node is unable to deliver a message to a child bridge node, it must send a DETACH.request packet, to the root node, which contains a detached node list that describes the lost subtree. The list contains all nodes in the routing table of the parent for which the lost bridge was the first upstream hop. All downstream bridge nodes in the path of the DETACH.request packet must adjust their routing tables by deleting entries which match those in the detached list. Detached node information is copied into a bridge's "detached node list".

Case 2 - The lost node is a terminal.

Since terminals can be mobile they can be lost often and must be notified quickly. When a parent node is unable to deliver a message to a terminal, it must generate a DETACH.request packet, with the terminal specified in the associated detached node list, and flood the packet throughout the network using the reliable flooding mechanism described above. Any bridge node, which receives the DETACH.request, adds the detached terminal to its internal detached node list.

A bridge node in the forward list does not forward an entry in the detached list of a DETACH.request if the DETACH.request came from an upstream node, and the upstream node is not the first hop in the routing table entry associated with the entry in the detached list. A DETACH.request is discarded if the detached list becomes empty.

Upstream bridges, which only have terminals nodes (i.e. do not have bridge nodes) as children, must convert a received DETACH.request to an unscheduled HELLO.response packet (without a forward list) and broadcast it immediately. Therefore, lost node information is flooded throughout the coverage area of the spanning tree. An awake unattached terminal should quickly discover that it has been detached and can then reattach.

Each entry in a bridge node's detached list is advertised in HELLO.response packets for MAX_HELLO_ LOST+2 HELLO times or until the bridge determines the terminal has reattached. The same entry can not be in a bridge's detached list for a hold-down time after it was deleted from the list.

Since a node assumes that it is detached if it does not receive a HELLO. response packet from its parent with MAX_HELLO_LOST HELLO time periods and detached node information is copied in HELLO.response packet for MAX_HELLO_LOST+2 HELLO time periods, a lost node is guaranteed that it will always discover when it becomes unattached within MAX_HELLO_LOST*HELLO_PERIOD seconds, in the worst case.

Unscheduled HELLO.response messages are generated in each bridge node if a new detached bridge is discovered and the time until the broadcast of the next scheduled HELLO.response packet is greater than HELLO_TRIGGER seconds.

Network Routing.

A routing table is maintained in each bridge node. The routing table has an entry for each known upstream node. (Downstream routing is defined by the structure of the spanning tree.)

An example routing table is shown below:

destination	port	first hop	age	child flag	delivery service	awake begin time	awake end time
hex 080A	1	hex 020A	0	false	null	null	null
hex 020A	1	hex 020A	0	true	null	null	null
hex 080B	1	hex 080B	1	true	1	15320	15330

In the example route table above, the bridge has three descendants - 080A, 020A, and 080B. 080A and 080B are terminal nodes and 020A is a bridge node. (Note that the node type can be determined by the address.) If the first hop field is the same as the destination field, then the (optional) child flag field is set to true. The delivery service, awake begin time, and awake end time fields only apply to child terminal nodes with a non-zero delivery service field.

To forward an upstream bridging layer packet, a bridge node looks up the entry associated with the destination address in the bridge header. If the entry does not exist, the packet is discarded. If the entry does exist, the MAC layer destination address is set to the first hop address in the route table. The MAC source address is set to the address of the bridge node. (The bridge header addresses remain unchanged.) The packet is passed to the MAC entity on the port specified in the table.

Route table entries are created or updated whenever a downstream unicast DATA, ATTACH, or ARP packet is received. If an entry does not exist for the bridging layer source address, an entry is created with the destination field set to the bridging layer source address. The fields in the (old or new) entry for the destination are modified as follows: 1) the first hop field set to the MAC layer source address, 2) the port field is set to identify the MAC entity which delivered the packet, 3) the age field is set to 0, and 4) if the destination and first hop fields are identical, the child flag field is set to true.

The age field for each entry is incremented at regular intervals. An entry's age field is reset to 0 whenever a packet is received from the destination associated with the entry. If no packets are received from the destination of an entry for ROUTE_TIMEOUT seconds, the entry is deleted from the route table.

Nodes can maintain their path in the network by sending an ATTACH.request packet to the root node once every ATTACH_TIMEOUT seconds, where ATTACH_TIMEOUT must be shorter than ROUTE_TIMEOUT. The ATTACH.request packet can be piggybacked on a higher-layer data packet, if the node is active.

If a DETACH.request packet is received from an upstream bridge node, then each entry in the route table, with a destination field which matches an entry in the packet's detached list, is deleted.

All nodes must maintain a **current_parent** structure. The structure has a network address field, a port field, and an age field. The age field is incremented once per HELLO period. If the count stored in the age field reaches MAX_HELLO_LOST, the current_parent structure is

re-initialized to null values, and the node becomes detached. The age field is reset to 0 whenever a scheduled or unscheduled HELLO.response packet is received from the node's parent. A downstream packets is forwarded by setting the MAC destination address to the current_parent.address and then passing the packet on the current_parent.port.

All nodes must maintain a **current_root** structure. The structure has an age field, a 48-bit long root ID field, a root alias field, and a root sequence field. The structure is initialized to all 0's (or null). The structure is checked, and possibly updated, each time a HELLO.response packet is received. If a node receives a HELLO.response packet which contains a long ID, alias, or root sequence number which is different than the associated value contained in the structure, then the node becomes unattached and loses its network address. The age field in the current_root structure is incremented periodically. If NETWORK_TIMEOUT minutes expire before a node receives a HELLO.response packet from any bridge node, then the current_parent structure is re-initialized and the node loses its address. The age field is reset to 0 whenever a HELLO.response packet is received from any bridge node and the root node has not changed.

Address Resolution and Maintenance.

Reverse Address Resolution Protocol (RARP) protocol.

An address server in the root node maintains network addressing information in an address table, distributes network addresses to requesting nodes, and resolves network addressing problems. Each entry in the address table contains a device type field, a network address field, a long ID field, an alias field, an in-use field, and an age field. Entries in the table are aged so that they can be reused after MAX_ADDRESS_LIFE minutes. Note that entries may remain in the table indefinitely. The age field in an entry is reset to 0 whenever a RARP.request or ATTACH.request packet is received from the node associated with the entry.

A separate sequential set of unique node identifiers is maintained for each device type. Each set begins with an identifier of 1 and ends with the maximum range for the device type. Lower node identifiers are distributed before higher identifiers.

A RARP.request packet can be used to: 1) acquire a network address from the address server, 2) change an existing 48-bit long ID in the address table, or 3) change an existing alias in the address table.

A node which does not yet have a unique 16-bit network address must request a 11-bit node ID from the address server. The node uses a 16 bit multicast address until a unique node ID is assigned. The low order 15 bits of the temporary address consist of the node's device type concatenated with an 11-bit node ID of all 1's. A RARP.request packet, containing the requesting node's unique 48-bit long ID and/or an alias, is sent to the address server by the requesting node. When a node requests a new address, the server first checks its address table to determine if the node already has a valid address. If the node doesn't already have an address, the server allocates the first available node identifier for the device type, to the node, if one is available. In either case, if an address is available, the server will set the network address field in the RARP packet to the allocated address and will set the return code bits to 0. If an address is not available, or an entry already exists in the address table with ambiguous identifiers, the address server will set the network address field to all 1's and will indicate the error in the return code field.

The long identifier and/or alias in a RARP.request packet matches an entry in the address table 1) if matching 48-bit IDs and aliases are present in both, 2) if both contain a matching alias and neither has a 48-bit ID , or 3) if both contain a matching 48-bit ID and neither has an alias. If either the 48-bit ID or alias in the request packet is also in the address table, and the above criteria are not met, an error is returned.

Occasionally, a node may want to change the association between a 48-bit ID and an alias. If the node simply requests a new address, and its old address has not expired, an error will be returned. The requesting node can override the error and force a change by setting the New Alias or New Long ID bits in the RARP.request operation field. A change operation causes a new address table entry to be created if neither the long ID or alias can be found in the address table.

A requesting node can specify that it does not have a 48-bit ID by not including it as a parameter or by including an ID of all 0's. An alias is required if there is no 48-bit ID. A requesting node can specify that it does not have an alias by not including it as a parameter, or by including an alias with a length of 0.

The address server will return a RARP.request packet to the requesting node as a RARP.response packet. If the node, which generated the RARP.request packet, does not receive a RARP.response packet within BRIDGE_TIMEOUT seconds, it must resend the RARP.request.

RARP hexadecimal 4-bit error codes.

0 = good.
1 = a node ID is not available.
2 = Duplicate alias. The alias specified in the RARP.request packet is already in the address table with a different long ID.
3 = Duplicate long ID. The long ID specified in the RARP.request packet is already in the address table with a different alias.
4 = Invalid device type.
F = Extended error code.

RARP routing.

Each bridge node maintains a RARP routing table which contains entries for upstream nodes which have recently sent a RARP.request packet to the root node.

An example RARP route table is shown below:

long ID	alias	port	first hop	network address	return code	age
hex 1003508A990C	null	1	hex 020A	hex FFFF	invalid	0
hex 1003508A920B	term2	1	hex FFFF	hex 080C	0	3

Whenever a RARP.request packet is received, an entry is created (or updated) in the RARP route table and the long ID and/or alias fields in the entry are set to the values specified in the request packet. **The node which initiated the request is defined by the l ng ID and/ r alias.** The long identifier and/or alias in a RARP.request packet matches an entry in the RARP route table 1) if matching 48-bit IDs and aliases are present in both, 2) if both contain a

matching alias and neither has a 48-bit ID , or 3) if both contain a matching 48-bit ID and neither has an alias. The return code is initialized to invalid to indicate that an associated RARP.response packet, destined for the node which originated the RARP.request, has not been received. The port field points to the port on which the RARP.request was received. The network address is set to the bridging layer source address of the RARP.request packet. (Normally, the node ID in the bridging layer source address will be all 1's, which is the default global node ID used before a unique network node ID is obtained. If a node is attempting to change its long ID or alias, then the network address may be unique.) The first hop field will be set to the MAC source address. The age field will be set to 0.

Normally, a bridge node will forward RARP.request packets to the root node on the port specified in the current_parent structure. However, if a bridge node receives a RARP.request packet, and 1) an entry for the node which initiated the request is already in the RARP route table, 2) the entry has a return code field which is valid, and 3) the network address field is not all 1's, then the bridge can simply return a RARP.response packet to the source node. The return code and age fields in the route table entry are not modified. The network address and return code fields in the RARP.response packets are set to the values contained in the RARP route table.

When a bridge node receives a RARP.response packet from the root node, it will update the return code and network address fields in the RARP route table entry for the node which initiated the request. RARP.response packets are forwarded on the port specified in the route table entry. The MAC destination address is set to the first hop address.

RARP route table entries are aged (quickly) so that older entries are discarded in RARP_TIMEOUT seconds.

Address Resolution Protocol (ARP) protocol.

A node can request the 16-bit network address of another node by sending an ARP.request packet to an address server in the root node. The ARP.request packet must contain either the 48-bit identifier or the alias of the target node, but not both. The address server returns the 16-bit network address of the target node in an ARP.response packet, if the target node exists in the server's address table. An address of all 1's and an error is returned if the target node is not in the address table.

ARP 4-bit hexadecimal error codes.

0 = good.
1 = long ID not found.
2 = alias not found.
F = extended error code.

Address Maintenance.

A node will lose its address if:

1) The root node changes (i.e. either a different ROOT ID or ROOT ID Sequence Number is detected in a HELLO.response packet).

2) It has not received an ATTACH.response packet from the root node, within an ADDRESS_TIMEOUT time period.

3) No network activity is detected within a NETWORK_TIMEOUT time period.

The root node can change because 1) the root node is lost or 2) a new root is chosen with the root selection protocol. All bridge nodes which detect the change must broadcast HELLO.response packets at scheduled times with the new ROOT ID and an infinite path cost for MAX_HELLO_LOST+2 HELLO times. If the root node is lost the new ROOT ID will be either a 48-bit long identifier of all 0's, if the old ROOT ID was a long identifier, or a null alias if the old ROOT ID was an alias. If the a new root is chosen the new ROOT ID will be either the 48-bit long identifier or alias of the new root node. Note that a new occurrence of the same root node can be always be detected because the ROOT ID sequence number will change.

A node can maintain its address by sending an ATTACH.request packet to the root node at least once per ADDRESS_TIMEOUT time period. Note that a node must send an ATTACH.request to the root at least once per ROUTE_TIMEOUT time period, to maintain its path to the root in the spanning tree; therefore no special logic is required for address maintenance. If the node is active it can simply piggyback the ATTACH.request on a higher-layer downstream data packet. The root node will return an ATTACH.response packet, and the node can reset its ADDRESS_TIMEOUT timer when the response packet is received.

HELLO Timing.

HELLO.response packets can be unscheduled or scheduled.

Unscheduled HELLO timing.

Unscheduled HELLO.response packets are broadcast in response to HELLO.request packets.

In addition, an unscheduled HELLO.response packet is generated in each bridge node if a new detached bridge is discovered and the time until the broadcast of the next scheduled HELLO.response packet is greater than HELLO_TRIGGER seconds.

Scheduled HELLO timing.

Each attached bridge node broadcasts one scheduled HELLO.response packet per HELLO_PERIOD seconds. Each HELLO_PERIOD time period is divided into AVG_HELLO_SLOTS slots, where each slot is HELLO_SLOT_SIZE seconds. Initially, a bridge node chooses a slot and broadcasts a HELLO.response packet. The HELLO.response packet contains a "seed" field which is used in a well-known "hello randomization algorithm" to determine the next hello slot and the next seed for the bridge node's next HELLO.response packet. The algorithm guarantees that a bridge node will randomly broadcast its next HELLO.response packet within a MIN_HELLO_PERIOD second to MAX_HELLO_PERIOD second time window after its last HELLO transmission.

Since significant cumulative delays in hello timing are prohibited, nodes can execute the hello randomization algorithm i times to determine the time (and seed) of the i-th successive scheduled HELLO.response packet from a bridge node. For example, a SLEEPING terminal node can initially synchronize on a HELLO.response packet from its parent. The terminal can calculate the time of a future HELLO.response packet from its parent and can power-down with an active timer interrupt set to wake it just before the broadcast of the HELLO.response packet is expected.

Contention delays incurred during the transmission of a HELLO.response packet are recorded in a "displacement" field in the packet. The displacement field specifies the transmission offset time, in hundredths of seconds, from the calculated transmission time. Note that errors, caused by rounding to the nearest displacement, can cause some drift in calculated transmission times. Therefore, nodes should resynchronize every time a HELLO.response packet is broadcast. If the displacement field, in a HELLO.response packet, is set to all 1's, then the transmission time of the packet does not coincide with a calculated hello time, and the packet should not be used for hello synchronization. The displacement is set to all 1's in unscheduled HELLO.response packets and is set to all 1's in scheduled HELLO.response packets whenever the packet can not be sent within MAX_HELLO_OFFSET milliseconds.

If an expected HELLO.response packet is not received at the calculated time, then the receiving node should wait for the maximum displacement time before assuming the HELLO packet was missed. The maximum displacement time can be calculated, in milliseconds, by multiplying the number of hello periods, since the last HELLO packet was received, by MAX_HELLO_OFFSET.

Note that HELLO slot boundaries for different bridge nodes are not necessarily aligned (i.e. as in slotted ALOHA). In fact, to increase randomization, bridge nodes should avoid slot alignment. A "HELLO slot" is simply the unit of time used in the randomization algorithm.

Default HELLO_PERIOD and HELLO_SLOT_SIZE values are set at compile time and are well-known by all nodes. Modified HELLO_PERIOD and HELLO_SLOT_SIZE values can be set by the root node and advertised throughout the network in HELLO.response packets.

The algorithm used for HELLO time calculation is defined by the "C" routines below. The routines assume that *last_hello_time* and *last_hello_seed* variables are maintained for each bridge node of interest. A bridge node should set the *last_hello_time* variable, associated with its HELLO.response packet broadcasts, to the last HELLO transmission time, adjusted to account for any displacement. For example, if a bridge node's HELLO transmission time is delayed approximately 20 milliseconds, then it should set the displacement field in the packet to 2 and sets its *last_hello_time* variable to the transmission time minus 20 milliseconds.

Two routines are shown below:

set_last_hello - is used to extract *last_hello_time* and *last_hello_seed* from a received HELLO. response packet. The last_hello_time variable is adjusted to account for the displacement field. The routine assumes that packets are accurately time-stamped (i.e. by the MAC layer) when they are received.

calc_next_hello - is used to calculate the time and seed of the next HELLO.response packet for the bridge specified by the address passed to the routine.

The routines rely on the address table functions - **Insert** and **Lookup**. **Insert** enters an address into a table, if it does not exist and returns the index. **Lookup** returns the index of an address, if it exists.

```
int set_last_hello_time(PACKET * hello_packet)
{
  offset=hello_packet->hello_seed & B_MASK_DISP;
  if (offset==B_MASK_DISP)
    return -1;  /* next hello time cannot be calculated, if the displacement is all 1's */

  /* else, determine the seed and hello time */
  i=Insert(hello_packet->mac_s_addr);
  last_hello_time[i]=hello_packet->receive_time  - ((TIME)(offset));
  last_hello_seed[i]=hello_packet->hello_seed >> 2;
  return 0;
}

int calc_next_hello(int address, TIME* next_time, int* next_seed)
{
  int next_slot;

  i=Lookup(address);
  if (i < 0) return -1;  /* next hello time cannot be calculated */

  *next_time=last_hello_time[i];
  *next_seed=last_hello_seed[i];
  while (*next_time <= current_time())
  {
    next_slot=((*next_seed+address) % HELLO_MOD_VAL) + HELLO_MOD_VAL;
    *next_time += next_slot * HELLO_SLOT_SIZE;
    *next_seed=((seed + 3) ^ 0x2A) & 0x3F;
  }
  return 0;
}
```

Sleeping terminal support.

The bridging layer provides a service which will store pending messages for SLEEPING terminals. Terminals request the service by setting the optional "delivery service" parameter (type hexadecimal 0E), in the optional parameter area in any bridging layer packet. If a child node does not set the parameter or sets it to 0 (the default value), then the parent bridge node will attempt to deliver packets to the child node immediately. The child node is assumed to be lost whenever a packet can not be delivered. If the parameter is set to 1, the source child node is assumed to be a SLEEPING terminal and the parent node will not attempt to deliver packets to the child node until the child notifies the parent that it is awake. If the parameter is set to 2, the parent bridge will attempt to deliver packets immediately, with a reduced retry count; if a packet can not be delivered, the parent will store the packet until the child node notifies the parent that it is awake (or a save time expires).

A child node can notify its parent node that it will be awake by setting an optional "awake time" parameter (type hexadecimal 0C) in any bridging layer packet. The awake time is specified in AWAKE_TIME_UNIT time units. A value of all 1's indicates forever. By default, the awake time period is assumed to follow the end of transmission of the packet containing the awake time parameter. Optionally, an offset can be specified with the optional "awake time offset" parameter (type hexadecimal 0D). The awake time offset parameter specifies an offset time at which time the terminal will wake up and stay awake for the time specified in the associated awake parameter. The offset is specified in AWAKE_TIME_UNIT time units and is added to the end-of-transmission time. This option allows a terminal to initiate a transaction, sleep for a statistically calculated time period, and then wake up to accept a response.

If a bridge node determines that a SLEEPING terminal is sleeping then it will store packets destined for the terminal for a number of HELLO periods, or until delivery to the child node is attempted. The number of HELLO periods that a packet will be saved is set to DEF_SAVE_CNT, by default, and can be set from 0 to MAX_SAVE_CNT by a child node with the optional "store message count" parameter (type hexadecimal 0F). Whenever a bridge node is storing a message for a terminal, the terminal's address will be included in the "pending message list" in successive HELLO.response packets broadcast by the bridge node. If the message can not be delivered within "store message count" HELLO periods then the bridge node will assume that the child terminal node is lost.

As an example, a SLEEPING terminal might notify its parent that it will be awake, in a time window when a reply message (i.e. from a host computer) is expected, by setting the awake time and awake time offset parameters in the packet which solicited the reply. If the reply message is not received when expected, the terminal can alternate between sleep periods and short awake time periods. An empty DATA.request packet containing a "wake time" parameter can be used to notify the parent of successive wake time periods. (Note that a parent node will generate an ATTACH.request packet with the ATTI bit set ON whenever a DATA.request packet is received from a terminal and a routing table entry does not exist for the terminal. Therefore, the empty DATA.request also ensures that the node is still attached to the network.) Whenever a SLEEPING terminal expects a reply message the terminal should wake up to receive all HELLO.response packets from its parent. If an expected message is not received during any of the awake periods, the terminal can simply continue to listen to scheduled HELLO.response packets. If a response message is expected and a threshold number (i.e. 1 or 2) of consecutive HELLO.response messages are missed, then the terminal should transmit a (empty) DATA.request packet to its parent to determine if it is still in range. Note that if the parent bridge fails to deliver a message to a child terminal then

either 1) the child will see its address in a "detached node list", or 2) will reattach to the network after a threshold number of HELLO.response messages are missed.

LLC implementation notes:

An LLC layer RR (receive-ready) packet, with the final bit set ON, should be transmitted by the terminal whenever the terminal attaches to the network. This ensures that higher layer data which may have been lost will be retransmitted immediately by the LLC entity at the remote end of the LLC connection.

Receive timeouts can be handled by the LLC entity in a terminal, if the bridging layer:

1) allows the LLC layer to pass an awake time window along with a data message.
2) provides a "request_message" service to the LLC layer. The request_message service causes the bridging layer to send an "awake_time" packet to the parent bridge node.
3) notifies the LLC layer whenever the terminal (re)attaches to the network.
4) provides a "net_monitor" service to the LLC layer. The net_monitor service causes the network layer to wake up for all HELLO packets from the parent bridge node, and causes the network layer to query the parent node (with an empty DATA.request packet) when a HELLO packet is missed.

An example algorithm for sending a message, for which a reply message is expected, using type 1 delivery service, is shown below:

```
awake_time_window->begin=minimum_data_delay;
awake_time_window->end=calculated_data_delay;
set_alarm(SHORT_TIMEOUT,
        awake_time_window->begin+awake_time_window->end+pause(0));
set_alarm(LLC_TIMEOUT, MAX_LLC_TIMEOUT);
net_send(destination, LLC_data, awake_time_window);*
net_monitor(TRUE);

while (waiting_for_response)
{
  event=wait(event_Q);
  if (event==SHORT_TIMEOUT && request_count++ < AWAKE_MAX)
  {
    request_message(awake_time);
    set_alarm(SHORT_TIMEOUT, pause(request_count));
  }
  else if (event==REATTACH)
  {
    awake_time_window->begin=0;
    awake_time_window->end=RR_TIMEOUT;
    request_count=AWAKE_MAX;
    reset_alarm(LLC_TIMEOUT, RR_TIMEOUT);
    net_send(destination, LLC_RR, awake_time_window);
  }
  else if (event==LLC_TIMEOUT && retry_count++ < LLC_MAX)
  {
    awake_time_window->begin=0;
    awake_time_window->end=RR_TIMEOUT;
    request_count=AWAKE_MAX;
    reset_alarm(LLC_TIMEOUT, RR_TIMEOUT);
    net_send(destination, LLC_RR, awake_time_window);
  }
  else
    waiting_for_response=FALSE;
}
net_monitor(FALSE);
set_alarm_off(SHORT_TIMEOUT);
set_alarm_off(LLC_TIMEOUT);
return event;
```

The while loop will terminate if 1) a reply message is received, 2) the maximum number of LLC timeouts is exceeded, or 3) a RR/RNR frame is received from the remote LLC entity. In the last case, the LLC state machine will either determine that the original LLC data frame must be retransmitted or that the LLC entity should continue to wait for a reply message.

*Note that the FINAL bit should be set OFF in the "LLC_data" frame so that an unnecessary RR frame will not be generated by the remote LLC entity.

Express Mail Label
No. EV 331533572US

Attorney Docket No.
14406US02

APPENDIX F

Title : Radio Frequency Local Area Network

Inventors: Meier, et al.

Attorney Docket No. 14406US02

MAC LAYER SPECIFICATION

Introduction.

This document, in conjunction with the SST network frame format specification, defines the hop-to-hop link protocol used on RS485 LAN links, spread spectrum radio links and ethernet links.

Functional Requirements and Capabilities.

The MAC layer:

- accepts frames from the bridging layer and passes frames to the physical layer for transmission.
- appends MAC layer framing bytes and CCITT-16 FCS bytes to transmitted frames.
- removes MAC layer framing bytes and FCS bytes from received frames.
- verifies the FCS bytes in received frames.
- filters out frames which do not belong to the SST network of the local device.
- filters out packets which are not directed to the local device.
- forwards packets to the bridging layer which are directly addressed to the local device, or are broadcast or multicast to the local device.
- regulates access to the communications channel on RS485 links and spread spectrum radio links.
- schedules lost unicast frames for retransmission.
- detects and discards duplicate back-to-back unicast MAC level data frames.

- provides device-to-device flow control.
- transparently fragments and reassembles bridging layer packets, which exceed the maximum MAC frame size.
- maintains and provides diagnostic statistics for higher layers.

Terminology and Definitions.

frame - MAC layer protocol data unit.

packet - bridge layer protocol data unit.

A **channel access algorithm** is used to regulate access to a communications channel. The implementation of a channel access algorithm is dependent on the link type:

A **p-persistent CSMA/CA** (carrier sense multiple access with collision avoidance) protocol is used to gain access to an **RS485 LAN**. The collision avoidance scheme gives channel access priority to the recipient of a unicast frame.

On lightly loaded spread spectrum radio links, a **non-persistent CSMA** algorithm is used to gain access to the communications channel.

On moderately to heavily loaded spread spectrum radio links, an **LBT/BP (listen-before-talk with busy pulse)** algorithm is used to gain access to the channel and minimize the effect of hidden nodes.

A **polling protocol** is used to restrict contention to **request-for-poll (RFP)** frames, thus minimizing contention for data frames.

A **CSMA slot** is defined as follows: Assume that station A has determined that the channel is idle at time **0** and immediately initiates a transmission. Then a CSMA slot is equal to the worst-case time, **t**, at which another station, B, is able to determine that the transmission from A is in progress. The actual components of the CSMA slot time are *<busy sense time> + <turnaround time> + <processing delay>*, where each of the components represents a worst-case value. The **busy sense time** is the time required to detect a frame in progress. The **turnaround time** is the time required by a half-duplex transceiver to switch from a receive state to a transmit state. The processing delay includes the hardware/software processing time required to initiate a transmission and MAC frame processing time.

An **LBT slot** is defined as the total time required to transmit an RFP frame plus the time required by the receiver to begin transmitting the response.

A **CA (collision avoidance) sequence** of frames consists of one or more frames which are sent following a single execution of the channel access algorithm.

An **interframe gap** is defined as the minimum idle time allowed between frames which are considered as a single CA sequence.

An **interpoll gap** is defined as the maximum time allowed between poll frames which belong to a single CA sequence.

A **return priority mechanism** allows frames which are transmitted between a pair of nodes to be grouped into a single CA sequence which is initiated with one execution of the channel access algorithm.

CSMA idle time is the minimum time that a potential transmitter must sense an idle channel before assuming the channel is idle. The CSMA idle time is greater than the interframe gap plus the CSMA slot size.

LBT idle time is the minimum time that a potential transmitter must sense an idle radio channel with hidden nodes before assuming the channel is idle. The LBT idle time is greater than the interpoll gap time plus the CSMA slot size.

A **bracket** of frames is defined as a group of one or more MAC-level frames which are associated with a single bridging layer unit of data.

MAC-level frames are categorized as either **request** or **poll** frames:

A **DATA frame** is a MAC-level **request** frame which is used to send higher-layer data to a receiver.

An **EOD (end-of-data) frame** is a MAC-level **request** frame which is sent as the last data frame in a bracket of one or more data frames. Note that a bracket of data frames may consist of a single EOD frame.

An **RFP (request-for-poll) frame** is a MAC-level **request** frame which is used to request polling from a receiver and to determine the SEQ state of the receiver.

An **ENQ (enquiry) frame** is a MAC-level **request** frame which is used to determine the SEQ state of a receiver and to determine if a node is within range.

A **POLL frame** is a MAC-level **poll** frame which is used to obtain a data frame from another node and to return the current SEQ state.

A **WFP (wait-for-poll) frame** is a MAC-level **poll** frame which is used to inform a requesting node that it is scheduled to be polled later and to return the current SEQ state.

A **CLEAR frame** is a MAC-level **poll** frame which is used to inform all listening nodes that the last frame in a bracket of frames has been received and to return a defined SEQ state.

A **REJECT frame** is a MAC-level **poll** frame which is used to return an undefined SEQ state or to indicate that a received request frame was invalid.

MAC control byte bit definitions.

Request or poll frame control byte bit definitions.

R/P (request/poll) bit.

The R/P bit is used to distinguish MAC layer request and poll frames. If the R/P bit is set OFF the frame is a request frame. If the R/P bit is set ON the frame is a poll frame.

SEQ bit.

The SEQ bit is used to sequence MAC layer data frames, modula 2. The SEQ field is used to detect and discard duplicate packets. A state machine which illustrates the use of the SEQ bit and the response ACC bit is shown below.

LAN ID bits.

The MAC frame belongs to the spanning tree specified by the LAN ID bits. The MAC entity discards frames which belong to spanning trees which are not in its LAN_ID_list. Note that LAN_ID_list is a parameter of the MAC_enable call.

Request control byte bit definitions.

DATA bit.

The DATA bit is used to distinguish control request frames from data request frames.

MORE bit.

In control request frames the MORE bit is used to distinguish RFP frames from ENQ frames.

In data request frames, the MORE bit is used to distinguish between DATA frames and EOD frames. The last frame sent in a bracket of data frames is always an EOD frame.

PRIORITY bit.

The PRIORITY bit indicates the priority of a higher layer message and is set as specified by the bridging layer, in the MAC_send call. The receiver simply passes the priority to the bridging layer. The PRIORITY bit value is the same for all frames which are associated with a bracket of frames

Poll control byte bit definitions.

MORE bit.

The MORE bit is used to distinguish POLL frames from CLEAR frames.*

WAIT bit.

The WAIT bit is used to distinguish POLL frames from WFP frames.* The receiver of a request frame can return a poll frame with the WAIT bit set ON in the associated poll frame to put the requesting node in a quiet state for WFP_TIMEOUT seconds. The requesting node must refrain from transmitting unicast frames to the receiver until the quiet period expires or a POLL frame is received from the receiver.

*A REJECT frame is specified by setting the MORE bit OFF and the WAIT bit ON.

Bridging Layer Interface Specification.

Each node in the network has a single bridging entity which invokes a MAC entity per port to send and receive messages on the port.

MAC layer services are provided with the following routines:

MAC_enable(port, LAN_ID_list)

MAC_set_address(port, net_address)

MAC_send(port, dest_net_address, buffer, control_flags, [mailbox], [queue])

control_flags bits (7-0)

bit 7	priority	0 = normal, 1 = high
bit 6	reserved	must be zero
bit 5	reserved	must be zero
bit 4	reserved	must be zero
bit 3	p_flag	1 = p-persistent
bit 2	reserved	must be zero
bit 1-0	LAN ID	(spanning tree ID)

length=**MAC_accept**(port, buffer, wait)

MAC_stop(port)

MAC_start(port)

MAC_disable(port)

MAC_enquiry(port, dest_net_address)

MAC_diagnostic(port, . . .)

Initially, the MAC entity attached to a port is in a DISABLED/OFF state. The bridging layer enables a MAC entity on a port by calling MAC_enable(port, LAN_ID_list), where LAN_ID_list defines the spanning trees to which the node can belong. MAC_enable changes the MAC entity state to ENABLED/ON.

The MAC entity uses a default multicast address consisting of the node type and a node identifier of all 1's, until the bridging layer assigns a specific network address to the MAC entity. The MAC_set_address call is provided for this purpose.

The bridging layer accepts messages from the MAC entity by issuing a MAC_accept call. The returned buffer includes the MAC header, but does not including media framing and CRC characters. The wait parameter can be used to suspend the caller for some length of time or until a message is received. The MAC entity must be capable of queueing messages until they are accepted by the bridging layer.

The bridging layer requests the MAC entity to transmit a bridging layer packet by issuing a call to MAC_send. Packets are grouped into a set of one or more MAC layer frames which, together, consitute a bracket. On radio ports, if the size of a bridging layer packet exceeds the maximum MAC frame length, then the packet is fragmented. A bracket normally contains a single data (EOD) frame on wired links. The MAC entity prefixes a MAC header to the beginning of each frame in a bracket before transmitting each frame. The MAC layer is also responsible for providing media framing, which includes a link-type-dependent synchronization preamble, start-of-frame delimiter, end-of-frame delimiter, and CRC-CCITT frame check sequence bytes for each frame. The c ntr l_flags parameter in the MAC_send call is used to 1) set the priority bit in the MAC header (priority), 2) to indicate if the buffer is being sent in response to a multicast bridging layer packet (p_flag), and 3) to set the LAN ID field in

the MAC header. The optional mailbox and queue parameters are mutually exclusive and are used for asynchronous calls.

The maximum size of a buffer passed to the MAC layer for transmission is MAX_PKT_SIZE.

The bridging layer can disable the MAC receiver by calling MAC_stop. The MAC entity is in an ENABLED/OFF state after a call to MAC_stop is issued. The bridging layer forces the MAC entity back into the ENABLED/ON state by calling MAC_send or MAC_start.

The bridging layer can disable the MAC entity and force it to the DISABLED/OFF state by calling MAC_disable.

MAC_enquiry can be use to determine if a destination node is within range.

MAC_diagnostic is used to retrieve diagnostic statistics from the MAC layer.

Link Interface Specification.

Frame/packet filtering.

When the MAC entity is in an ENABLED/ON state it is continuously listening on its assigned port. The MAC entity receives all MAC layer frames. Frames which do not pass a CRC-CCITT check are invalid and are discarded. Valid data frames are reassembled into a complete packet which is posted to the bridging entity if:

1) The LAN ID in the MAC header is among those contained in the LAN ID list passed to the MAC entity in the MAC_enable call, and

2) The destination address in the MAC header a) is equal to the network address of the local node, or b) is an acceptable multicast or broadcast address.

The high-order multicast bit is set ON in all multicast or broadcast frames. A multicast or broadcast frame is accepted if the node type specifies a group to which the local node belongs and either a) the node identifer is all 1's, or b) the node identifer is equal to the identifier of the local node. A response is never required when the multicast bit is set ON.

The default network address used when the MAC entity is first enabled consists of the multicast node type concatenated with a node identifier of all 1's. For example, the default address for a bridge is hexidecimal A7FF. The bridging layer is responsible for obtaining a network address and assigning it to the MAC entity on the port.

Channel access.

A return priority mechanism is used to group MAC layer request and poll frames into a single CA sequence. A channel access algorithm is executed to gain access to the channel before the first frame in a CA sequence is transmitted. All other frames in a CA sequence may be sent without executing the channel access algorithm. The idle time between frames which belong to a single CA sequence must be less than the maximum interframe gap time.

On wired links, the **CSMA/CA** algorithm forces nodes to detect an idle channel for a **CSMA idle time** which exceeds the interframe gap time before initiating a CA sequence.

On radio links "hidden nodes" can cause throughput to be significantly degraded on spread spectrum radio links. Under lightly loaded conditions, a CSMA channel access algorithm allows nodes to access the radio channel immediately after detecting an idle channel. Under moderate to heavily loaded conditions, the **LBT/BP** algorithm forces nodes to detect an idle radio channel for an **LBT idle time** which exceeds the **interpoll gap** time, before accessing the channel. By listening for longer than the interpoll gap time, a node will detect a conversation in progress, if both involved nodes are in range or only one node is in range and the other node is hidden. Limiting the time between frames in a CA sequence to a short fixed interval, essentially provides a **busy-pulse** signal which spans the coverage area of both nodes involved in a conversation.

A CA sequence of frames begins with the transmission of a request or poll frame, following an execution of the channel access algorithm. Possible successive frames in a CA sequence are:

1) Any poll frame sent in response to a unicast request frame.
2) A DATA or EOD frame sent in response to a POLL frame.
3) A bridge node can "piggyback" a second frame onto a transmitted broadcast, multicast, WFP, CLEAR, or REJECT frame, by transmitting the second frame within the interframe gap time.

Message Bracketing.

The size of packets which are passed to the MAC layer by the bridging layer must be less than, or equal to, MAX_PKT_SIZE, where MAX_PKT_SIZE specifies the total length of the packet, including bridging and data-link header characters.

Packets which are larger than MAX_FRAME_SIZE must be fragmented, by the MAC entity, to insure that the interpoll gap time is constant. The fragmented frames are transmitted as a **bracket** with the MORE bit set OFF in the last frame to mark the end of the bracket. Frames which belong to a single bracket are reassembled by the MAC entity in the receiver before the packet is posted to the bridging layer in the receiver. If the entire bracket is not received successfully, then all other frames in the bracket are discarded by the receiver. Note that the maximum number of data frames in a bracket is the ceiling of MAX_PKT_SIZE / MAX_FRAME_SIZE.

MAX_FRAME_SIZE does not include characters added at the MAC level. MAX_FRAME_SIZE on the 192K bps spread spectrum radio link is limited by the interpoll gap time.

On a wired links with low error rates, MAX_FRAME_SIZE is set so that a bracket is generally limited to a single EOD frame.

A bracket of frames may be transmitted in one or more CA sequences, where a channel access algorithm is used to gain access to the link for each CA sequence. A transmitter initiates the transmission of a bracket of frames by sending either an RFP frame or an EOD frame to a receiver. If a receiver is not busy, the receiver will respond to RFP and DATA frames with a POLL frame, which solicits the next DATA frame and implicitly acknowledges the previous frame. A receiver responds to an EOD frame with a CLEAR frame. If a receiver is busy or does not have a buffer, then the receiver may respond to RFP, DATA or EOD frames with a WFP frame.

Example transmit/receive sequences.

The following sequences illustrate typical transmission sequences, in the absence of errors, for sending a bracket of data frames from a transmitter to a receiver.

Sequence 1:

```
RFP    ---------------------->
       <---------------------- POLL
EOD    ---------------------->
       <---------------------- CLEAR
```

Sequence 2:

```
EOD    ---------------------->
       <---------------------- CLEAR
```

Sequence 3:

```
RFP    ---------------------->
       <---------------------- POLL
DATA   ---------------------->
                .
                .
       <---------------------- POLL
EOD    ---------------------->
       <---------------------- CLEAR
```

Sequence 4:

```
RFP    ---------------------->
       <---------------------- WFP
          (pause)
       <---------------------- POLL
DATA   ---------------------->
                .
                .
       <---------------------- POLL
EOD    ---------------------->
       <---------------------- CLEAR
```

Sequence 5:

```
ENQ    ---------------------->
       <---------------------- CLEAR
```

Sequences 3 through 4 can only occur on radio links. Note that, in sequence 2, an EOD frame is sent immediately, without sending a prior RFP frame. It is possible to skip the transmission of an RFP frame if the transmitter "remembers" the SEQ state of the receiver. This facility is limited to the transmission of short EOD frames on wired links. Sequence 5 can be used to determine the SEQ state of a receiver or to determine if a node is within range.

Recovery.

The node which initiates a bracket of frames (i.e. the transmitter) is responsible for recovery until the first POLL frame is received.

The receiver is responsible for polling the transmitter as soon as an RFP frame is received and assumes responsibility for recovery at that point.

It is possible for both the transmitter and receiver to be in contention to recover a lost frame (i.e. RFP or DATA) if the first POLL frame is lost. The contention is resolved with a random backoff algorithm.

If a CLEAR frame is lost and the polling node which sent the CLEAR frame is responsible for recovery, then the requesting node which initiated the bracket can not determine if the link was lost or the CLEAR frame was lost. The requesting node must send an *ENQ* frame to determine which case holds.

Transmit and Receive State Machine (SM) Specifications.

No state machine is required for multicast and broadcast frames. Multicast and broadcast frames can be transmitted whenever the channel is available. Received multicast or broadcast frames are simply discarded or posted to the bridging layer.

State Machines for Bracket Transmission.

Bracket Transmit State Machine.

State descriptions:

IDLE - The state machine is idle and is waiting for a bracket of frames to transmit.

READY - The state machine has a bracket of 1 or more frames to transmit and is waiting to acquire the channel.*

S_RFP - The state machine has sent an RFP frame and is waiting for a POLL frame.

S_DATA - The state machine has sent a DATA frame and is waiting for a POLL frame.

S_EOD - The state machine has sent an EOD frame after receiving a POLL frame and is waiting for a CLEAR frame.

RDY_WAIT - The state machine has received a WFP frame and is waiting for a POLL frame (or timeout).

The following states only apply to transmissions on a wired link which are not initiated with a request for polling.

READY2 - The state machine has a single short frame to transmit, is waiting to acquire a wired link, and the SEQ state of the receiver is known.

S_EOD_2 - The state machine has sent an unsolicited EOD frame is waiting for a CLEAR frame.

* There is an automatic and immediate transition from the READY state to the READY2 state if all of the following conditions are true: 1) the communciations channel is a wired link; 2) the SEQ state of the receiver is known, and 3) the bracket to transmit consists of a single EOD frame which is less than MAX_SHORT_FRAME_SIZE in length.

Timers:

A RSP_TIMEOUT receive timer is started when a) an RFP frame is transmitted, 2) an ENQ frame is transmitted, and 3) on wired links, when an EOD frame is sent without first sending an RFP frame. The timeout value is larger than interframe gap time plus the time required to transmit a POLL or CLEAR frame. If the RSP_TIMEOUT timer expires before an expected response is received, a retry counter is incremented and the request frame is retransmitted, if the retry count has not been exceeded.

A POLL_TIMEOUT receive timer is started whenever a DATA or EOD frame is transmitted following an RFP frame. The timeout value is larger than the time required for the maximum number of poll retry attempts. The MAC layer returns an error to the bridging layer if this timer expires before an expected poll frame is received. Note that the receiver is responsible for recovery when this timer is running.

A WFP_TIMEOUT timer is started whenever a WFP frame is received. The RDY_WAIT state ends when this timer expires or a POLL frame is received.

The state machine must maintain a "current pointer" variable which points to the current frame, in a bracket of frames, to be transmitted. The current pointer is advanced if, and only if, a POLL for the next frame in the bracket is received. If more than one transition is specified when a POLL frame is received, the state of the current pointer determines which transition should be taken.

Retransmission logic and SEQ state maintainence are specified in the section which describes state machines for frame SEQ control.

The first state table below specifies transitions for bracket transmissions which are initiated with a polling request. The second table specifies transitions for non-polled frame transmission (i.e. a single short EOD frame on a wired link). Note that there are transitions between the tables.

state	event	action	next state
IDLE	A bracket of frames is passed to the state machine	Reset retry count; execute channel access algorithm	READY
READY	Channel acquired	Increment retry count; send RFP frame; start RSP_TIMEOUT receive timer	S_RFP
S_RFP	RSP_TIMEOUT timer expires and max. retry count exceeded	Return error	IDLE
	RSP_TIMEOUT timer expires	Execute channel access algorithm	READY
	POLL received	Send current DATA frame; start POLL_TIMEOUT receive timer	S_DATA
	POLL received	Send current EOD frame; start POLL_TIMEOUT receive timer	S_EOD
	WFP received	Start WFP_TIMEOUT timer	RDY_WAIT
S_DATA	POLL_TIMEOUT timer expires	Return error	IDLE
	POLL received	Advance current pointer if frame was accepted; send current DATA frame; start POLL_TIMEOUT receive timer	S_DATA
	POLL received	Advance current pointer if frame was accepted; reset retry count; send current EOD frame; start POLL_TIMEOUT receive timer	S_EOD
S_EOD	POLL_TIMEOUT timer expires	Reset retry count; execute channel access algorithm	RDY_ENQ
	POLL received	Retransmit EOD frame; start POLL_TIMEOUT receive timer	S_EOD
	CLEAR received; EOD frame not accepted	Return error (invalid transition)	IDLE
	CLEAR received; EOD frame accepted	Return good	IDLE
RDY_ENQ	Channel acquired	Increment retry count; send ENQ; start RSP_TIMEOUT timer	S_ENQ
S_ENQ	RSP_TIMEOUT timer expires and max. retry count exceeded	Return error	IDLE
	RSP_TIMEOUT timer expires	Execute channel access algorithm	RDY_ENQ
	CLEAR received; EOD frame not accepted	Return error (invalid transition)	IDLE
	CLEAR received; EOD frame accepted	Return good	IDLE
RDY_WAIT	WFP received	Reset WFP_TIMEOUT timer	RDY_WAIT
	POLL received	Send current DATA frame; start POLL_TIMEOUT timer	S_DATA
	POLL received	Send current EOD frame; start POLL_TIMEOUT timer	S_EOD
	WFP_TIMEOUT timer expires	Reset retry count; execute channel acess algorithm	READY

state	event	action	next state
READY›	Channel acquired	Increment retry count; send EOD frame with RESET on; start RSP_TIMEOUT receive timer	S_EOD2
S_EOD›	RSP_TIMEOUT timer expires and max. retry count exceeded	Return error	IDLE
	RSP_TIMEOUT timer expires	Execute channel access algorithm	READY›
	WFP received	Start WFP_TIMEOUT timer	RDY_WAIT
	CLEAR received; EOD frame not accepted	Return error (invalid transition)	IDLE
	CLEAR received; EOD frame accepted	Return good	IDLE

Bracket Receive State Machine.

The receive state machine assumes that invalid frames and frames not directed to the local node are discarded and do not affect state transitions. Multicast and broadcast frames are simply posted to the bridging entity, if a buffer is available, and do not affect state transitions.

State descriptions:

IDLE_LISTEN - The receiver is not receiving a bracket of frames.

BUSY - The receiver has sent a POLL frame and is waiting for the next frame in a bracket.

BUSY_WAIT - The receiver is waiting for a buffer to become free.

Timers:

A RSP_TIMEOUT receive timer is started when a POLL frame is transmitted. The timeout value is larger than interframe gap time plus the time required to transmit a DATA frame. If the RSP_TIMEOUT timer expires before an expected response is received, a retry counter is incremented and the POLL frame is retransmitted, if the retry count has not been exceeded.

The receiver must maintain a **poll_queue** which is a FIFO list of all terminals which have requested polling. (Note that a separate queue can be used for high priority requests.) Entries in the queue are aged so that they are discarded after WFP_TIMEOUT seconds. The entry at the front of the queue is considered **active**; all other entries in the queue are denoted as **queued**. Nodes which are not active or queued are denoted as **inactive**. Note that there is no active node in the IDLE_LISTEN state.

A SEQ state variable is cached for all nodes which have recently transmitted valid data frames. The SEQ state variable is updated as specified in the section which describes state machines for frame SEQcontrol.

Only one bracket may be in progress at a time. The receiver must reserve enough buffers for an entire bracket of frames before sending a POLL frame in response to an RFP frame. This ensures that the entire bracket will be accepted.

"**Flush**" deletes the entry for a node from the poll_queue, if it exists, and frees any buffers allocated to frames received from the node.

state	event	action	next state
IDLE_LISTEN	RFP received; buffers available	Send POLL; insert node in poll_queue; reset retry count; start RSP_TIMEOUT timer	BUSY
	RFP received; buffers not available	Send WFP; insert node in poll_queue	BUSY_WAIT
	EOD received; buffer available	Send CLEAR; post complete packet (if accepted); flush	IDLE_LISTEN
	EOD received; buffer not available	Send WFP; insert node in poll_queue	BUSY_WAIT
	ENQ received; entry for source node is in the SEQ state table	Send CLEAR	IDLE_LISTEN
	ENQ received; no entry for source node in the SEQ state table	Send REJECT	IDLE_LISTEN
	DATA received	Send REJECT	IDLE_LISTEN
BUSY	max. retries exceeded	Flush; delete SEQ state table entry	IDLE_LISTEN
	receive timeout	Increment retry count; execute channel access algorithm; acquire channel; resend POLL; start RSP_TIMEOUT timer	BUSY
	DATA received from active node	Reset retry count; send next POLL; start RSP_TIMEOUT timer	BUSY
	DATA received from inactive or queued node	Send REJECT	BUSY
	EOD received from active node; no queued nodes	Send CLEAR; reassemble and post complete packet; flush	IDLE_LISTEN
	EOD received from active node; 1 or more queued nodes	Send CLEAR; reassemble and post complete packet; flush; reset retry count; send POLL to node at front of poll_queue; start RSP_TIMEOUT timer	BUSY
	EOD received from inactive or queued node; buffer available	Send CLEAR; post complete packet (if accepted); flush	BUSY
	EOD received from inactive or queued node; buffer not available	Send WFP; insert node in poll_queue	BUSY
	ENQ received from inactive or queued node; entry for source node is in the SEQ state table	Send CLEAR; flush	BUSY
	ENQ received from inactive node or queued node; no entry for source node in the SEQ state table	Send REJECT; flush	BUSY

	ENQ received from active node; no queued nodes	Send REJECT ; flush; delete SEQ state table entry	IDLE_LISTEN
	ENQ received from active node; 1 or more queued nodes	Send REJECT; flush; delete SEQ state table entry; reset retry count; send POLL to node at head of poll_queue; start RSP_TIMEOUT timer	BUSY
	RFP received from inactive or queued node	Send WFP; insert node in poll_queue	BUSY
	RFP received from active node	Send WFP if poll_queue is not empty; flush; insert node in poll_queue; reset retry count; send POLL to node at head of poll_queue; start RSP_TIMEOUT timer	BUSY
BUSY_WAIT	buffer becomes available	Send POLL to node at head of poll_queue; start RSP_TIMEOUT timer	BUSY
	DATA received	Send REJECT	BUSY_WAIT
	EOD received	Send WFP; insert node in poll_queue	BUSY_WAIT
	ENQ received; entry for source node is in the SEQ state table	Send CLEAR; flush	BUSY_WAIT
	ENQ received; no entry for source node in the SEQ state table	Send REJECT; flush	BUSY_WAIT
	ENQ received from active node; no queued nodes	Send REJECT ; flush	IDLE_LISTEN
	RFP received	Send WFP; insert node in poll_queue	BUSY_WAIT
	buffer becomes available	Send POLL to active node; start RSP_TIMEOUT timer	BUSY

State Machines for Frame SEQ Control.

All unicast MAC data frames are sequenced with a 1-bit sequence number (SEQ). The sequence number is used to detect lost data frames and duplicate data frames.

The MAC entity in each node must maintain transmit and receive SEQ state tables for unicast messages. The receive SEQ state table contains an entry for each active MAC source node. The transmit SEQ state table contains an entry for each active destination node. Each entry consists of a 1-bit SEQ state variable and a network address. **Only unicast command frames affect state table entries.** As a rule, a receive table entry should be discarded before the counterpart transmit table entry (i.e. in another node) is discarded. Receive SEQ state table entries need only be kept long enough to ensure that retransmitted duplicates are not mistaken for valid frames. This implies that receive table entries must be kept for a period longer than the maximum transmit retry time for a single frame. An entry in the transmit SEQ state table can be kept until the space is required for a new entry. Strict state timing is not required because a transmitter, without a table entry for a potential receiver, can determine the state of a receiver, with an RFP frame, before transmitting data frames. Also, note that the MAC layer does not provide a reliable service. Lost frames and duplicates are detected by higher layers.

The SEQ state machin descriptions below specify how entries in the SEQ state tables are maintained. Note that "poll" is used to denote any poll frame (i.e. POLL, WFP, CLEAR, or REJECT) and "data" is used to denote any data frame (i.e. DATA or EOD). Specific frame types are always in upper case.

Receive SEQ Control State Machine.

SEQ State descriptions:

ACCEPT_0 - the receiver expects the next DATA or EOD packet to have a SEQ number of 0.

ACCEPT_1 - the receiver expects the next DATA or EOD packet to have a SEQ number of 1.

ACCEPT_ANY - the receiver will accept a DATA or EOD packet with a SEQ number of 0 or 1.

The MAC receiver caches receive SEQ state variables for active external source nodes. The variable can be set to one of three states listed above. A state of ACCEPT_ANY applies to all nodes which do not have entries in the receiver's SEQ state table. The receiver sets the SEQ bit in a poll frame to denote the next frame that the receiver expects.

State	Event	Action	Next State
ACCEPT_ANY	Receive RFP/ENQ	Return poll 0	ACCEPT_ANY
	Receive data 0	Accept frame and return poll 1	ACCEPT_1
	Receive data 1	Accept frame and return poll 0	ACCEPT_0
	Receive data 0, no buffer	Discard frame and return WFP 0	ACCEPT_0
	Receive data 1, no buffer	Discard frame and return WFP 1	ACCEPT_1
ACCEPT_0	Receive RFP/ENQ	Return poll 0	ACCEPT_0
	Receive data 0	Accept frame and return poll 1	ACCEPT_1
	Receive data 1	Discard frame and return poll 0	ACCEPT 0
	Receive data 0, no buffer	Discard frame and return WFP 0	ACCEPT_0
	State timeout	Delete state entry	ACCEPT_ ANY
ACCEPT_1	Receive RFP/ENQ	Return poll 1	ACCEPT_1
	Receive data 1	Accept frame and return poll 0	ACCEPT_0
	Receive data 0	Discard frame and return poll 1	ACCEPT_1
	Receive data 1, no buffer	Discard frame and return WFP 1	ACCEPT_1
	State timeout	Delete state entry	ACCEPT_ANY

Transmit SEQ Control State Machine.

SEQ State descriptions:

SEND_0 - the transmitter sends the current data frame with a SEQ number of 0 and expects a POLL or CLEAR with a SEQ number of 1.

SEND_1 - the transmitter sends the current data frame with a SEQ number of 1 and expects a POLL or CLEAR with a SEQ number of 0.

UNKNOWN - the transmitter must send an RFP or ENQ frame to determine the SEQ state of the receiver.

The MAC transmitter maintains a transmit SEQ state variable per external node. The transmitter sets the SEQ field in DATA and EOD frames to the value of the transmit SEQ state variable. The state variable can be in one of the three states listed above. The UNKNOWN state applies to all nodes which do not have entries in the transmitter's state table. If the state is UNKNOWN, the transmitter must send an RFP or ENQ frame, to determine the SEQ state of the receiver, before sending a data frame.

The SEQ field in a poll frame denotes the next data frame expected. Each time a poll frame is received, the transmit SEQ state variable, associated with the source of the poll frame, is set to the value of the poll frame's SEQ field. A "current pointer" points to the current data frame in a bracket of data frames. The current pointer is advanced if the current data frame has been transmitted with a SEQ field value of 0(1) and a poll frame is received with a SEQ field value of 1(0).

On radio links, the SEQ state is set to UNKOWN as soon as the transmission of a bracket of frames is completed.

Network Constants.

WFP_TIMEOUT = 1 second, is the time that a node remains in a quiet state waiting for a POLL frame after a WFP frame is received.

MAX_PKT_SIZE = 800 bytes, is the maximum size of a bridging layer packet including bridging header characters.

R_MAX_FRAME_SIZE = 100 bytes, is the maximum size of a MAC layer frame on the spread spectrum radio link, not including MAC header and framing characters.

W_MAX_FRAME_SIZE = MAX_PKT_SIZE, is the maximum size of a MAC layer frame on the RS485 LAN, not including MAC header and framing characters.

W_MAX_SHORT_FRAME_SIZE = 200 bytes, is the maximum size of a MAC layer frame which can be transmitted without first sending a RFP frame on the RS485 LAN.

W_SLOT_SIZE = 50 microseconds, is the CSMA slot size for the RS485 LAN.

W_INTERFRAME_GAP = 200 microseconds, is the maximum interframe gap time for the RS485 LAN.

W_IDLE_TIME = W_INTERFRAME_GAP + W_SLOT_SIZE + 50 microseconds, is the CSMA idle time on the RS485 LAN.

R_SLOT_SIZE = 1000 microseconds, is the **LBT slot** size on the spread spectrum radio link.

R_INTERPOLL_GAP = 5000 microseconds, is the **interpoll gap** time on the spread spectrum radio link.

R_IDLE_TIME = R_INTERPOLL_GAP, is the **LBT idle time** on the spread spectrum radio link.

Channel access algorithms.

The CSMA/CA channel access algorithm used on the RS485 LAN differs from the LBT algorithm for radio links because of the hidden terminal factor in the radio network.

CSMA/CA summary:

The p-persistent CSMA/CA algorithm forces all nodes to detect an idle channel for one CSMA idle time unit, where a CSMA idle time unit is greater than the interframe gap time, before the channel is considered free. If a node initially detects a free channel, it can transmit immediately. If a node detects a busy channel, it listens to the channel until it becomes free. When the channel becomes free, at that point, time is divided into p CSMA slots. The node selects on of the p slots, say i, at random. If the channel is idle for the first i-1 slots, then the node will transmit in slot i. If the channel becomes busy in one of the first i-1 slots, the process is repeated. If an expected response is not received, a node will choose a number, i, between 1 and p, and will delay for i CSMA slots before re-executing the CSMA algorithm to retransmit. The number of backoff slots, p, is given as an increasing function of the number of missed responses and busy channel detections.

LBT/BP summary:

The LBT algorithm functions as a pure CSMA algorithm when the channel is lightly loaded. A channel is allowed to transmit as soon as an idle channel is detected. CSMA is never used for retransmissions. When the channel is moderately to heavily loaded, the LBT algorithm forces all nodes to detect an idle channel for at least one LBT idle time unit, where an LBT idle time unit is greater than the interpoll gap time, before the channel is considered free. If a node initially detects a free channel, it can transmit immediately. If a node detects a busy channel, it listens to the channel until it becomes free. When the channel becomes free, at that point, time is divided into p LBT slots. The node selects on of the p slots, say i, at random. If the channel is idle for the first i-1 slots, then the node will transmit in slot i. If the channel becomes busy in one of the first i-1 slots, the process is repeated. If an expected response is not received, a node will choose a number, i, between 1 and p, and will delay for i LBT slots before re-executing the LBT algorithm to retransmit. The number of backoff slots, p, is given as an increasing function of the number of missed responses and busy channel detections.

The CSMA/CA algorithm for the RS485 LAN, and the LBT/BP algorithm for spread spectrum radio links are both shown in pseudo-code below.

LBT/BP algorithm for a transmitter on the radio network.

```
BACKOFF_INIT     = 20;
R_RSP_TIMEOUT    = R_INTERPOLL_GAP;
MAX_TX_TRIES     = 20;
MAX_IDLE_TRIES   = 50;
```

rf, rg - functions which return a maximum backoff number based on the input parameter.

Wait for a MAC_send call.

```
if p_flag is non-zero then
begin
  select a random number, i, between 0 and BACKOFF_INIT;
  SLOT_OFFSET =i*R_SLOT_SIZE;
end
else
  SLOT_OFFSET=0 ;

TX_RETRIES=0;
IDLE_RETRIES=0;

while TX_RETRIES < MAX_TX_TRIES and IDLE_RETRIES < MAX_IDLE_TRIES and not OK do
begin
  OK=False;
  detect an idle channel for SLOT_OFFSET+R_IDLE_TIME time units;
  SLOT_OFFSET=0;
  if channel is idle then
  begin
    send_frame;
    if a return priority response is expected then
    begin
      wait for response or R_RSP_TIMEOUT timeout;
      if a valid response has been received then
        OK=True;
      else (assume a collision has occurred)
      begin
        TX_RETRIES=TX_RETRIES+1;
        select a random number, j, between 0 and rf(TX_RETRIES);
        SLOT_OFFSET=j*R_SLOT_SIZE;
      end
    end
  end
  else (the channel is not idle)
  begin
    wait until the channel is idle;
    IDLE_RETRIES=IDLE_RETRIES+1;
    select a random number, k, between 0 and rg(IDLE_RETRIES);
    SLOT_OFFSET =k*R_SLOT_SIZE;
  end
```

CSMA/CA algorithm for a transmitter on the RS485 LAN.

```
BACKOFF_INIT     = 20;
W_RSP_TIMEOUT    =1 millisecond;
MAX_TX_TRIES     = 20;
MAX_IDLE_TRIES   = 50;
```

wf, and *wg* - functions which return a maximum backoff number based on the input parameter.

Wait for a MAC_send call.

```
if p_flag is non-zero then
begin
  select a random number, i, between 0 and BACKOFF_INIT;
  SLOT_OFFSET =i·W_SLOT_SIZE;
end
else
  SLOT_OFFSET=0 ;

TX_RETRIES=0;
IDLE_RETRIES=0;

while TX_RETRIES < MAX_TX_TRIES and IDLE_RETRIES < MAX_IDLE_TRIES and not OK do
begin
  OK=False;
  detect an idle channel for SLOT_OFFSET+W_IDLE_TIME time units;
  SLOT_OFFSET=0;
  if channel is idle then
  begin
    send_frame;
    if a return priority response is expected then
    begin
      wait for response or W_RSP_TIMEOUT timeout;
      if a valid response has been received then
        OK=True;
      else (assume a collision has occurred)
      begin
        TX_RETRIES=TX_RETRIES+1;
        select a random number, j, between 0 and wf(TX_RETRIES);
        SLOT_OFFSET=j·W_SLOT_SIZE;
      end
    end
  end
  else (the channel is not idle)
  begin
    wait until the channel is idle;
    IDLE_RETRIES=IDLE_RETRIES+1;
    select a random number, k, between 0 and wg(IDLE_RETRIES);
    SLOT_OFFSET =k·W_SLOT_SIZE;
  end
```

Media Access Framing.

All frames must be preceded with a preamble which is necessary for bit and character synchronization.

SST RF frame preamble: 6 to 8 flag bytes (hex 7E).
SST RF frame trailer: 1 flag byte (hex 7E)
RS485 LAN frame preamble: 1 to 6 flag bytes (hex 7E).

The preamble is defined in addition to the flag start-of-frame delimiter. For example, frames on the RS485 LAN must begin with 2 to 7 flag bytes. The preamble is required for receiver synchronization.

Line Turnaround Timing.

SST RF turnaround timing.

A transmitting device on an RF link must be in receive mode and begin training less than 75 microseconds after shifting the last bit of the trailing flag byte of a unicast frame onto the link to guarantee that a response will not be lost.

A receiver on the RF link must wait at least 75 microseconds, after receiving the end-of-frame flag byte in a unicast command frame, before transmitting the last 5 bytes in the preamble of the response frame. Fast devices can begin transmitting sooner by adding extra flag byte(s) to the beginning of the response frame. Since each character, at 192 Kbps, occupies 41.7 microseconds, an 8-byte preamble is sufficient to remove the entire 75 microsecond wait time constraint. The first byte in the preamble must be sent within 200 microseconds.

RS485 LAN turnaround timing.

A transmitting device on the wired RS485 LAN must be in receive mode less than 75 microseconds after shifting the last bit of the end-of-frame flag byte onto the link in a unicast frame. Note that this time allows approximately 35 microseconds for transmitting a 16-bit abort sequence and 40 microseconds for the line turnaround.

A receiver on the wired backbone must wait at least 75 microseconds, after receiving the end-of-frame flag byte in a unicast frame, before transmitting the last byte in the preamble of the response frame. Since each character, at 460.8 Kbps, occupies 17.4 microseconds, a 5-byte preamble is sufficienct to remove the entire 75 microsecond wait time constraint. The first byte in the preamble must be sent within 200 microseconds.

Express Mail Label
No. EV 331533572US

Attorney Docket No.
14406US02

APPENDIX G

Title : Radio Frequency Local Area Network

Inventors: Meier, et al.

Attorney Docket No. 14406US02

PHYSICAL LAYER SPECIFICATION

Introduction.

This document describes the physical characteristics of communications links used by devices in the SST network. Medium access techniques mentioned in this document are further defined in the MAC layer specification.

Communication Link Types.

Six physical communication link types have been defined for devices in the SST network:

1) Direct-sequenced Spread Spectrum Radio.
2) IEEE 802.3.
3) RS485 LAN.
4) RS485 Passive Bus.
5) RS232 Point-to-point.
6) V.35

(In addition, multiple RS422 links can be used to control and communicate with UHF base stations which coexist in the SST network; however the UHF base stations are not considered as part of the SST network.)

Only the first three link types are used for communications within the SST network. The passive bus link type is used for off-line batch communications to Norand devices. RS232 is used for communications with various devices, including host computers, printers, etc. RS232, V.35 and 802.3 can all be used to provide a controller-to-host-computer link.

Direct -sequenced Spread Sprectrum Interface.

Spread spectrum radio provides wireless communications for online terminals in a mobile environment. Spread spectrum radio links are also used to extend the range of wired radio base stations. The wireless base station option facilitates ease of installation and changes in base station topology.

The spread spectrum channel frequency is 902 to 928 mhz.

IEEE 802.3 (ethernet) Interface.

The IEEE 802.3 interface provides a LAN industry standard backbone channel for the Norand SST network. Controllers and/or base stations can be distributed, as required, on an 802.3 LAN to provide radio coverage for spread spectrum terminals. A Norand proprietary wired backbone is not required. In addition, an 802.3 LAN can provide the link to a host computer.

802.3 support is implemented with an optional LAN interface module, available on controllers and high-end base stations. Each device which supports the 802.3 interface has a twisted-pair compatible RJ45 connector, and a thin coaxial cable compatible BNC connector. Support for thick coaxial cable networks must be provided with an off-the-shelf thin-to-thick adaptor module.

The IEEE 802.3 standard is defined in the ISO 8802-3 standard.

RS485 LAN (type 1) Interface.

The RS485 LAN or type 1 interface provides an inexpensive medium-speed wired backbone network for devices in the SST system. The RS485 LAN is designed to minimize delay for sporadic transaction-oriented communications in a, primarily, online environment; however, efficient batch communications is also supported on the backbone.

Type 1 network characteristics are listed below:

- LAN architecture.
- EIA RS485 physical link.
- 460.8 Kbps link speed.
- Bit-synchronous data link protocol.
- FM0 data/clock encoding scheme.
- Access to the link is through p-persistent CSMA hardware and software.
- Link terminations may be AC coupled.
- Up to 32 devices are allowed per LAN link segment before a repeater is required.
- Link segment lengths up to 1000 feet are allowed before a repeater is required.
- A collision avoidance mechanism is provided for acknowledgments and batch traffic.

The type 1 LAN standard is hierarchical in the sense that all type 1 devices must also support the type 2 interface described below.

The 460.8 Kbps standard was chosen because:

- 460.8 is an even multiple of conventional baud rates (i.e. 48 x 9600).
- The 230.4 Apple Localtalk standard can be achieved by simply dividing the rate by two.
- A Zilog 8 Mhz 8530 SCC with a 7.3728 Mhz PCLK can achieve the 460.8 Kbps rate (FM0 encoding or asynchronous) by gating PCLK to the receive clock input. The 7.3728 Mhz PCLK is required to achieve conventional baud rates, up to 115.2 Kbps, with the 8530 baud rate generator. Many off-the-shelf communications products are implemented with the 8530 chip.

RS485 Passive Bus (type 2) Interface.

The RS485 passive bus interface is designed to reduce contention for batch communications with devices in the SST network and existing DSD terminals. The type 2 interface provides a passive bus for multidropped terminals. A master controller device regulates access to the bus by slave terminal devices.

Devices on the RS485 passive bus are not "online" in the SST network. Access to a host computer is through an RS485 port on a controller device.

Type 2 network characteristics are listed below:

- Bus architecture.
- EIA RS485 physical link.
- 115.2 Kbps link speed.
- Access to the link is directed by a "master" controller.
- The access method is essentially TDM. A single device owns part of the channel for an entire file transfer session.
- NRZ asynchronous data encoding.
- Up to 32 devices per link segment are allowed before a repeater is required.
- Link segment lengths up to 1000 feet are allowed before a repeater is required.
- Link terminations are DC coupled.

The 115.2 Kbps asynchronous standard was chosen because:

- 115.2 is an even multiple of conventional baud rates (i.e. 12 x 9600).
- The 115.2 Kbps speed can be implemented with an interrupt driven or polled UART on slower devices.
- 4000 DSD terminals are capable of 115.2 Kbps asynchronous RS485 communications.

RS232 Point-to-point (type 3) Interface.

The RS232 type 3 interface is required to interface with off-the-shelf RS232 devices, host computers, and existing Norand RS232 equipment (i.e. printers). RS232 devices must support link speeds up to 38.4 Kbps. Type 1 devices should support bit-synchronous RS232 communications. All devices must support asynchronous RS232 communications.

Exampl 1.

Figure 1.1 illustrates the use of some of the physical link types described above.

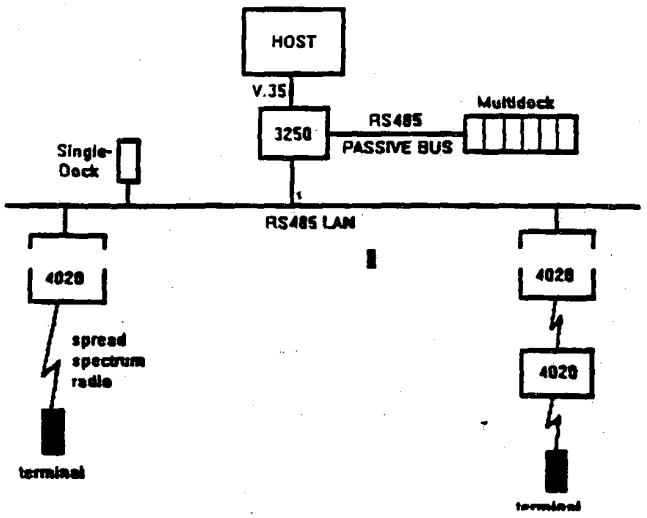

figure 1.1

Example 2.

Figure 1.2 illustrates how an 802.3 LAN can be used as the backbone in an SST network. Note that it is possible to intermix 802.3 and RS485 links in the same SST network.

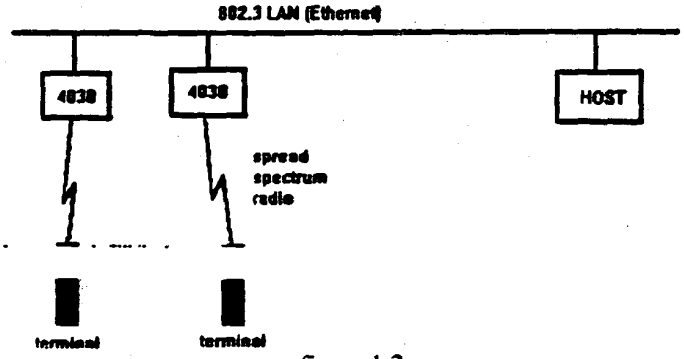

figure 1.2